

- *Historical Review of OS/2 Porting*
- *OS/2 Warp for the PowerPC*

March/April 1995
Volume 7 Number 2

PRODUCT REVIEW

**VisPro/C,
VisPro/C++**

OS/2 Developer

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT

**Programming for
Portability**

**Converting OS/2
Applications to
32 Bits**

**Integrating 32-Bit Code
with 16-Bit Code**

**The Dos and Don'ts
of Porting Applications**

**Buyer's Guide:
Application
Migration Tools**

**MIGRATING
APPLICATIONS
AT**

WARP

SPEED

U.S. \$9.95 Vol. 7 No. 2

© 1995 by IBM Corp. All rights reserved. No part of this publication may be reproduced without prior written permission from IBM Corp.

WHEN I WANT ADVICE ABOUT OBJECT-ORIENTED
PROGRAMMING, I'LL ASK AN EXPERT.



SO, ASK US.

You already know reusable class libraries promise a dramatic increase in productivity and efficiency for professional application developers. You also know class libraries created with one C++ compiler can't link with code created by another C++ compiler. And that libraries written in one language can't be used with client code written in another language.

But here's something you might not know: MetaWare and IBM are changing all that.

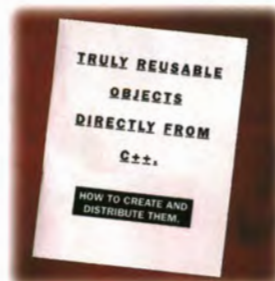
Out With The Old. In With The New. The tight binary coupling that exists between object-oriented class libraries and client code means that changes to either—no matter how small—require a complete recompilation not only of the class itself, but of all the client modules as well.

But with IBM's System Object Model (SOM) and MetaWare's High C/C++ DirectToSOM compiler for OS/2, developers can break that tight binary coupling and extend the advantages of procedure libraries to object-oriented technology.

Pretty amazing, right? And right now, you think this is where our pitch to sell you a compiler comes in. But you're wrong.

Information Only. And It's Free. We know what you really want is information. That way you can make up your own mind about the benefits of SOM and the DirectToSOM High C/C++ compiler for OS/2. We've prepared a white paper that will help. It's called Truly Reusable Objects Directly From C++: How to Create and Distribute Them. It's yours just for the asking.

To receive your free information, call, fax, or e-mail MetaWare at the numbers shown below. Once you review the facts for yourself, you'll understand why MetaWare and IBM mean "objects made easy."



CALL 408 429 6382
FAX 408 429 9273
OR
E-MAIL TECHSALES@METAWARE.COM
FOR YOUR FREE COPY.
TODAY.

 **MetaWare®**
OBJECTS MADE EASY.

The Suite Way to Build Portable Applications

NEW!

Is building applications your job? Then life just got easier thanks to the zApp® Developer's Suite for Windows. The zApp Developer's Suite is a set of highly integrated C++ development tools designed to help you transform the blueprints in your mind into commercial quality applications—quickly and easily. And best of all, applications built using the zApp Developer's Suite are portable to fourteen different platforms!

The zApp Developer's Suite consists of zApp, the award-winning portable C++ application framework; zApp Factory™, a fully visual application designer and code generator; and the zApp Interface Pack, a collection of high-level visual objects for the zApp environment. All of these tools are highly integrated to provide maximum ease of use and flexibility.

Rapid Application Development.

Introducing an exciting new visual technology that lets you drag and drop a wide assortment of objects like toolbars, tables, and 3D dialogs; define their characteristics; and build interfaces of any



work, and the zApp Interface Pack, so you have all of their power at your disposal — toolbars, table objects, advanced graphics — in all, over 300 object classes of power just waiting to be tomorrow's best-selling application.

Portability and More.

When you're done building your application, *then* you can decide what platforms you want to support! Applications built using the zApp Developer's Suite are single-source portable to fourteen different platforms. By simply recompiling, your application will run natively on Windows, Win32 (Windows NT, Windows 95, and NT on the DEC Alpha), OS/2®, DOS Text, DOS Graphics and seven X/Motif platforms: IBM RS/6000 AIX, HP HPUX 9.x, SGI IRIX 5.2, SCO UNIX, Sun Solaris 2.x, Sun SunOS 4.1.x, Sun Solaris x86.

complexity; all in one powerful but easy to use environment. With the click of a button, you can engage a powerful test mode which lets you interact with your application, seeing it exactly like your end user will see it: letting you fill in dialogs, pull down menus, etc. When you are pleased with the look and feel of your application, fully commented C++ source code is only a mouse click away, thanks to the zApp Developer's Suite's code generation capabilities.

Object-oriented Power.

The best news is that this development environment sits on top of zApp, the industry leading C++ application frame-

Free Demo.

Sound impossible? Well, if seeing is believing for you, call **1-800-346-6275**, ask for our free demo disk, and get a glimpse of what the future has to offer.



zApp and Inmark are registered trademarks of Inmark Development Corporation. zApp Factory is a trademark of Inmark Development Corporation. OS/2 is a registered trademark of IBM. All other trademarks are the property of their respective owners.



INMARK

2065 Landings Drive, Mountain View, CA 94043
T: 800-346-6275 or 415-691-9000 F: 415-691-9099

In the U.K. and Scandinavia, call PTS/Software Plus at +44 (0) 928 579900

In France, call PTS/Software Plus at (05) 908194
In Germany, call ESM Software at 07022-9256-0
In Italy, call Silicon Valley On-Line at (049) 654221
In Australia, call Micro Way at (03) 580-1333
BBS: 415-691-9990 • Internet: info@inmark.com
CompuServe: GO INMARK

Fast Visual Application Development for OS/2 and DB2



If you're looking for fast and easy application development for OS/2, then take a look at the award-winning Watcom VX•REXX visual development environment. VX•REXX lets you build applications to exploit the graphical user interface, multi-threading, and multi-processing power of OS/2. VX•REXX Client/Server Edition gives you the added power to access DB2 or other database systems, manipulate the

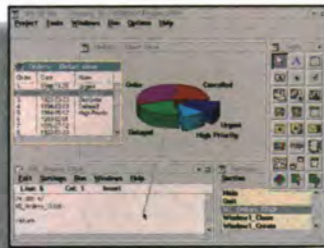
data, and chart the results at lightning speed.

"We like VX•REXX. Using it for development feels like driving a Porsche: it's fast, it's compact, everything's in the right place, and it makes us look good, too."
Peter Coffee, PC WEEK

Designed to Meet Your Needs.

Watcom VX•REXX combines a project management facility, visual designer and an interactive debugger to deliver a highly productive visual development environment. The Client/Server Edition includes additional powerful objects so you can rapidly create rich GUI database applications. You can create OS/2 client applications which connect to DB2/2 or DB2/6000. Use IBM's DRDA support on OS/2 to access DB2 for MVS, DB2/400 for AS/400, and DB2/VSE and VM (SQL/DS) for VM and VSE. Also supported are Watcom SQL and ODBC-enabled databases.

"Overall, this edition of VX•REXX for OS/2 is an outstanding visual client/server development platform." Nicholas Petreley, InfoWorld



Point. Click. And Presto!

To create an application you draw user interface objects, customize their properties using standard OS/2 notebooks, and define their event code using powerful drag-and-drop programming. To add database access just draw a query object, visually design a SQL query, press OK and presto— your window is automatically populated with objects that are bound to your query to display, update and search your data.

"Drag-and-drop nirvana." Nicholas Petreley, InfoWorld

Give Your Data a Whole New Image.

Energize your applications by displaying your data in a 3D chart. The Client/Server Edition gives you more than a dozen chart types to choose from, along with over 150 display options. You also get complete support for run-time events so you can bring new drama to your data by making your chart interactive.

"VX•REXX is a must buy." Jacques Surveyer, ComputerWorld

Standard or Client/Server Edition— Which one is for you?

To start creating powerful OS/2 GUI applications right away, order your copy of Watcom VX•REXX Standard Edition for just....\$99*

Or, to start creating rich client/server database applications, order Watcom VX•REXX Client/Server Edition for just.....\$299*

- Over 2 dozen objects, including CUA'91 containers, notebooks, pop-up menus and more
- Integration and control of existing applications through DDE, keystrokes or REXX API's
- Easy to learn event-driven programming model with complete on-line documentation
- Support for professional multi-threaded, multi-windowed and drag-and-drop enabled applications
- Code reusability through section and file sharing
- Graphically create CUA'91 Presentation Manager objects, quickly customize their properties, and easily attach REXX procedures
- Package your application as an EXE or PM macro for royalty-free distribution



1-800-265-4555

Watcom

A Powersoft Company

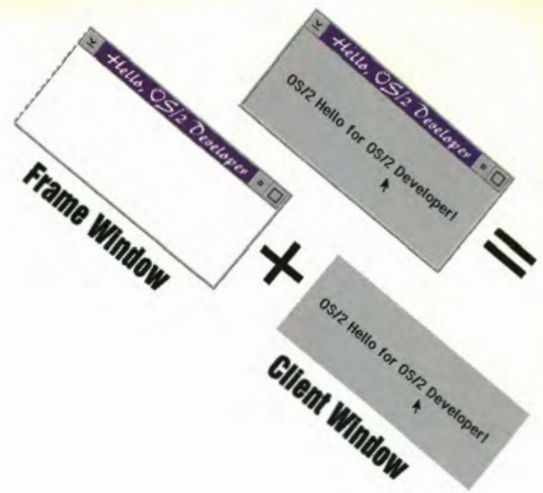
Watcom International 415 Phillip Street, Waterloo, Ontario, Canada N2L 3K2 Tel. (519) 886-3700 Fax (519) 747-4971 *Prices and specifications are subject to change without notice. Price does not include freight and taxes where applicable. Prices quoted in US dollars.
† ODBC drivers are available from INTERSOLV, Inc. Watcom, the Lightning Device, and VX•REXX are trademarks of Watcom International Corporation. Other trademarks are properties of their respective owners. ©Copyright 1994 Watcom International Corporation.

Circle Reader Service Number 9

OS/2 Developer

MARCH/APRIL 1995

THE MAGAZINE FOR ADVANCED SOFTWARE DEVELOPMENT



EDITOR'S COMMENTS

Getting from There to Here 5



OS/2 TOOLBOX

Product Reviewed: VisPro/C, VisPro/C++ 8



GUI CORNER

This Isn't the Gutenberg Press 16



PROGRAMMING INSIDER

Power Applications 22



APPLICATION MIGRATION

Eicon Technology Plays it Smart 26

By Steve Evans

Taking the Plunge: Converting an OS/2 Application to 32 Bits 33

By Dean Roddey

Warp and Windows: How Similar, How Different 42

By Bernie Thompson

Porting Applications to OS/2: a Historical Review of OS/2 49

By Derrel Blain

Don't Throw Away Those 16-Bit Object Libraries 56

By Alan Auerbach

OS/2 Programming for the Windows Guru 66

By Lou Miranda



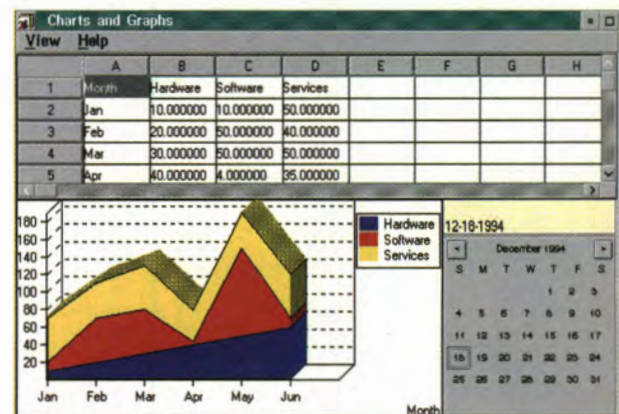
SOFTWARE TOOLS

Application Migration Buyer's Guide 74



PRODUCT WATCH

New Products for OS/2 77





Programmer's Paradise®

Call for a
FREE
Programmer's
Paradise
OS/2 Catalog!



Gammatech REXX SuperSet/2 by Softouch Systems

The SuperSet/2 gives REXX the power of C by providing interfaces to LAN Server, NetBIOS, TCP/IP and Communications Manager/2, and by providing access to low level system facilities, including processes, threads, semaphores, names, pipes, and VIO commands. The SuperSet/2 complements visual REXX programming packages. The 600+ page manual fully documents over 300 functions in 7 DLLs ready to supercharge your REXX!

Ours: \$69

FAXcetera #: 3621-0002

Product Code: GTGR33100

key 01MA95

Visual SlickEdit for OS/2 by MicroEdge Inc.

Visual SlickEdit—The intelligent programmer's editor. This high-powered programmer's editor is completely customizable with the speed to ignite your productivity. Save time using Brief, Emacs, or VI emulations. Take off with Visual SlickEdit's GUI interface, built-in dialog editor, and C-style macro language. Features include SmartPaste™, compiler error processing, syntax color-coding, expansion and indenting, and an on-line manual. 30 day risk-free trial!



Ours: \$199*

FAXcetera #: 1997-0002

* Special price good through March 31, 1995.

Product Code: MEVSE3100

key 03MA95



Watcom VX•Rexx Client/Server 2.1 by Watcom

A visual development environment for OS/2. Powerful connection, query and chart objects allow you to access several databases, manipulate data and chart the results quickly and easily. Features include drag-and-drop programming; bound controls; professional multi-threaded, multi-windowed and drag-and-drop enabled application development.

Ours: \$275

FAXcetera #: 1683-0022

Product Code: WATVC3100

key 05MA95

EZRAID PRO & EZRAID LITE by PRO Engineering, Inc.

The OS/2 software solution that provides the same full powered RAID benefits found in hardware systems worldwide. With its unique technology, EZRAID PRO increases system performance, simplifies data management and ensures complete fault tolerance. It supports spanning, RAID levels 0, 1, 4 and 5; is compatible with SCSI, IDE and ESDI; and is ready for Warp and LAN Server 4.0.

New
EZRAID PRO
Version!



Lite
PRO
FAXcetera #:

Ours: \$188
Ours: \$764

1016-5101

Product Code: PEELO3100

Product Code: PEEPO3100

key 07MA95

RimStar™ Programmer's Editor by RimStar Technology, Inc.



Features: 32-bit GUI, Syntax Coloring, ANSI "C" macro language, "C" Source Browser, unlimited redo/undo, no limits on number of files, windows, or line length. Emulates BRIEF plus many other editors. Compile/jump to error, hex editing, smart indenting, bookmarks, completely configurable keyboard mappings, WF/2 enabled and much more. Windows & NT versions available.

Ours: \$259

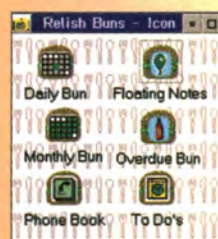
FAXcetera #: 1013-0901

Product Code: RSRSP3100

key 02MA95

Relish by Sundial Systems

Calendar, to do's, and phone book are fully integrated for total reliability, yet also independent for maximum flexibility. Schedule realistically for any time and duration, double-booking as necessary; categorize commitments; repeat events; print schedules; run programs; dial calls; prioritize to do's. Drag-drop, desktop objects ("Buns"), keyword searching, expert system for times/dates. Easy, convenient, CUA compliant, OS/2 Certified.



Ours: \$89

FAXcetera #: 1015-9901

Product Code: SSREL3100

key 04MA95



c-tree Plus® by FairCom

DOS • WINDOWS • NT • UNIX • OS/2 • SUN • RS6000 • HP9000 • MAC • QNX • BANYAN • SCO. This well known, highly portable data management package has become established as the tool of choice for commercial development. Offering unprecedented data control, choose from direct low level access, ISAM level, or SQL access with the FairCom Server. Single User, MultiUser, or optional Client/Server, ANSI Standard. Full Source. No Royalties.

Ours: \$495

FAXcetera #: 1381-0004

Product Code: FACT+3100

key 06MA95

To order call: 800-445-7899

Corporate Developer Division: 800 441-1511

FAX: 908 389-9227

International: 908 389-9228

Customer Service: 908 389-9229

Programmer's Paradise Deutschland:

Tel.: 08121/79073 • FAX: 08121/76566

Programmer's Paradise Italia:

Tel.: 39-2-967-00409 • FAX: 39-2-967-02855

Programmer's Paradise UK:

Tel.: 0161 7284177 • FAX: 0161 7284017

For more information on the

products featured on these pages call

FAXcetera®: (908) 389-8173

Programmer's Paradise®

1163 Shrewsbury Avenue
Shrewsbury, NJ 07702-4321

* Prices subject to change without notice.
• Call for shipping charges/return policies.

EDITOR*Dick Conklin***MANAGING EDITOR***Marta Dils***EDITORIAL ASSISTANT***Deborah Sommers***EDITORIAL ARTIST***Peter Altenberg***VICE PRESIDENT/****GROUP DIRECTOR***Regina Starr Ridley***PUBLISHER***Peter Westerman***ADVERTISING SALES***Yvonne Labat**(415) 905-2353**Christian O'Brien**(212) 626-2322**Kristin Morgan**(212) 626-2498***MARKETING MANAGER***Susan McDonald***MARKETING ASSISTANT***Larra Dutton***ADVERTISING PRODUCTION****COORDINATOR***Glenn Sadin***CIRCULATION DIRECTOR***John Rockwell***CIRCULATION MANAGER***Stephanie Blake***SUBSCRIPTION INQUIRIES***U.S.: (800) WANT-OS2**Foreign: (708) 647-5960***CHAIRMAN OF THE BOARD***Graham J.S. Wilson***PRESIDENT/CEO***Marshall W. Freeman***EXECUTIVE VICE PRESIDENT***Thomas L. Kemp***VICE PRESIDENT, PRODUCTION***Andy Mickus***VICE PRESIDENT, CIRCULATION***Jerry Okabe***BY DICK CONKLIN**

Editor's Comments

Getting from There to Here

I think that this issue is a real "keeper" because of its relevance to so many of our readers. Many of you have developed for multiple platforms or migrated from Windows to OS/2, facing many choices along the way. Should you use software tools or hand-code the changes? Is it better to use a cross-API library or a code converter? Do you program for portability or performance, exploiting the unique advantages of each platform? Should you migrate to 32 bits or stay at 16? And what about object libraries?

This issue features six articles by Windows and OS/2 platform experts focusing on the relative similarities and differences between the two operating systems and on writing applications for each. Author Derrel Blain reviews the conversion and porting tools for Windows to OS/2 migration, including One Up's Source Migration Analysis Reporting Toolset (SMART). Steve Evans of Eicon Technology used SMART to port a 3270/5250 terminal emulator to OS/2. Dean Roddey upgraded his company's clinical information system to a full 32-bit OS/2 application. Author Lou Miranda tells Windows gurus about OS/2 dos and don'ts. IBM OS/2 insiders Bernie Thompson and Alan Auerbach offer a gold mine of tips and techniques for both Windows and OS/2 developers.

I expect that these articles will provoke some interesting discussions in our CompuServe forum (GO OS2DF2, OS/2 Developer Mag section), which has shown an increase in activity lately. This is a great place to meet the authors, download source code examples, and get some questions answered. Drop in and say hello.

Dick Conklin

*Dick Conklin*

Contact	CompuServe	Internet
Dick Conklin (Editorial)	76711,1005, or OS2DF2 Forum	os2mag @vnet.ibm.com
Miller Freeman (Circulation)		mjohnson @mfi.com
Peter Westerman (Publisher)	76307,1054	pwesterman @mfi.com
Marta Dils (Managing Editor)	75152,134	mdils @mfi.com

OS/2 Developer: Vol. 7, No. 2, March/April 1995

Subscription information: U.S.: One year (six issues), \$39.95. Canada, Mexico, and international surface mail, add \$16 per year for postage. Canadian GST no. 124513185. International air mail, add \$30 per year for postage. Foreign orders must be accompanied by payments in U.S. funds. For new orders and customer service, call (800) WANT-OS2 (800-926-8672) or (708) 647-5960 (fax: 708-647-5972). IBM employees and branch office customers can subscribe to *OS/2 Developer* through IBM Mechanicsburg's Systems Library Services (SLS) using *OS/2 Developer*'s order number: G362-0001. POSTMASTER: Please send address changes to *OS/2 Developer*, P.O. Box 1079, Skokie, IL 60076-8079. Allow eight weeks for subscriptions to begin. *OS/2 Developer* (ISSN 1073-0729) is published bimonthly by Miller Freeman Inc., 600 Harrison Street, San Francisco, CA 94107, (415) 905-2200. Application to mail at second-class postage rates is pending at San Francisco, CA, and additional mailing offices.

OS/2 Developer has applied for BPA membership.

© Copyright 1995 by Miller Freeman Inc. PRINTED IN THE U.S.A.

un

Miller Freeman, Inc.
A MEMBER OF THE UNITED NEWSPAPERS GROUP

A man in a dark suit, white shirt, and patterned tie is smiling and clapping his hands. In the background, a red sports car is visible.

MAZDA'S I.S. CHIEF

CAN'T WAIT TO

GET WARPED.



OS/2 is in its third rev, so it's solid, stable, and mature.

Mike Anzis is the I.S. man behind the wheel of Mazda's computers. And OS/2® Warp is about to make his life easier.

As Mike puts it: "We use OS/2 on our headquarters' client/server systems. It's also installed in our 900 dealerships.

But until now, we haven't been able to get

OS/2 everywhere we need it - on laptops in the field. OS/2 Warp changes all that." OS/2 Warp is the 32-bit, multitasking, Windows™-friendly way to run a computer. With simple installation and



A toolbar gets users into their applications quickly and easily.

proven reliability, OS/2 Warp is a total computing solution that performs ruggedly at every level.

From a basic 4MB laptop to a client/server system, the OS/2 family now scales an even

wider variety of PC platforms.



The BonusPak gives mobile users the applications they need.

And OS/2 Warp is a real communicator.

With fax, Internet e-mail, and desktop conferencing, there

isn't an easier way to keep those out on the road in the loop.

OS/2 Warp also offers Mike Anzis rock-solid reliability. "I know from years of experience with OS/2, I can trust it to keep performing. Now I can enjoy this peace of mind at every level."

OS/2 Warp is available for under \$90. To get warped, stop by your local software dealer, or call 1 800 3 IBM-OS2. Ask for a free demo disk.

The new 32-bit, multitasking, multimedia, Internet-accessed, crash-protected, Windows-friendly, totally cool way to run your computer. **OS/2® WARP**



OS/2 Warp is available from your software dealer. It's also available from IBM for \$89 by calling 1 800 3 IBM-OS2.

Reseller prices may vary. OS/2 Warp consists of OS/2 version 3 and BonusPak. IBM, Operating System/2 and OS/2 are registered trademarks of the International Business Machines Corporation. Crash Protection and the OS/2 logo are trademarks of IBM. Windows is a trademark of Microsoft Corporation. ©1994 IBM Corp. All rights reserved.

Circle Reader Service Number 5



This issue, our new OS/2 Toolbox column reviews two visual application development tools from HockWare Inc. **By GUY SCHARF**

Product Reviewed: VisPro/C, VisPro/C++



Guy Scharf

VisPro/C and VisPro/C++ are new visual application development tools from HockWare Inc. for creating native OS/2 Presentation Manager applications in C and C++. These two products bring the power of drag-and-drop programming for C and C++ to OS/2 Presentation Manager. Finally, you can create C and C++ applications as easily as you can REXX applications! VisPro/C generates C code to use the OS/2 Presentation Manager API, while VisPro/C++ uses the IBM C Set++ ICLUI class library.

Both products support the standard OS/2 Presentation Manager controls, including notebooks, containers, sliders, and spin buttons, but not the MMPM/2 controls. Add-on objects provide formatted entry fields, a spreadsheet, business graphics, and other functions. The Database Designer allows you to reverse engineer DB2/2 databases, design databases using entity-relationship diagrams, and build SQL statements for your application.

Both products support only the IBM C Set and C Set++ compilers. DB2/2 must be installed to use the Database Designer. Each product requires about 2.5 megabytes of disk space.

I tested VisPro/C and VisPro/C++ both as standalone application develop-

ment tools and using IBM DB2/2 single-user and client/server database systems. In this review, I will use "VisPro" to refer to features common to both products.

USING VISPRO/C AND VISPRO/C++

Starting a new application in VisPro/C or VisPro/C++ is as simple as tearing a form off the Projects template. This creates a new project folder for your application. The project folder contains a template for forms, an initial Main form, empty Source and SubProcs folders, and skeleton files for compiler and linker options and for application resources.

The VisPro products use a form metaphor. Every window of the application is represented in VisPro by a form. While designing the application, you place objects such as buttons and entry fields on the form. Click on an object with the right mouse button, and a pop-up menu allows you to open the object's settings, arrange the object with respect to other objects, and open an editor for a specific event.

The settings notebook supports most OS/2 Presentation Manager styles. The styles not included are ones for which accompanying code support is not provided. For example, `LS_OWNERDRAW` is not included for the listbox. You can



A PROGRAMMING TOOL THAT WON'T SCREW YOUR HEAD UP

Splat! There goes another masterpiece. It was going to be a work of art, a monument to elegance and flexibility. It was going to be your finest creation. What happened? It got beaten into unrecognisable garbage by some clumsy GUI builder. As for the controls, they ended up as subtle as a flying sledgehammer thanks to a resource editor with an attitude.

This kind of mental torment could turn a decent living programmer into a serial killer. Somebody out there had better come up with a solution.

We just did. It's called Prominare, a professional's programming tool for GUI creation that has all the answers.

Take a look at the code it generates and you'll find it looks very familiar. It's exactly the way you would have written it yourself. If that sounds a bit scary, don't worry, it's real friendly. Define your naming convention and Prominare will stick to it to the letter. Prominare generates code exactly the way you want it. Your way.

If your looking to create custom controls. Prominare is a designers dream. Use it as the resource editor and you'll find that you have enough options to give you total freedom. It will also handle all major



object libraries and happily deal with PEN and multimedia. It handles all versions of OS/2 and there's no problem with understanding your old resources or Windows resources either.

Apart from being friendly, Prominare is also intelligent. When you make modifications to an application you won't be hindered by unnecessary generation phases, because it only regenerates the parts that have been modified. If you wish to make a manual modification Prominare will honour and obey it.

Now here's the best bit. We've produced a shareware version called Prominare Lite that's freely available on the Internet. Just help yourself to it and see how it beats the stink out of anything else. Then get your head around this question. If the shareware version is this good, what will the full version do?



Prominare

Tel: +1 (416) 363 2292 Fax: +1 (416) 363 6157
Tel: +44 (117) 972 8500 Fax: +44 (117) 972 8600
e-mail: designer@interlog.com

anonymous ftp: gold.interlog.com:/pub/prominare



add initialization code to the form to add these styles during execution using the OS/2 API, however, and you can add processing code as required to implement the style.

Initially, each form is a blank canvas on which you can place objects by using menu or toolbar selection or by dragging a tool from the tool palette. You can complete the graphical design before coding

processing logic, or you can intermix design and coding.

Before creating any events referring to an object, you should assign a symbolic name to the object. Lacking a name, VisPro will use the numeric value assigned to the object throughout any code it generates or any code that you create using drag-and-drop programming. It's hard to change the num-

ber to a name later, as you have to locate any references to the number and change them manually. VisPro does not propagate name changes through your event handling code.

Each object has a set of events to which it can respond. An event is some action signaled by the object, such as a WM_CONTROL message. On the pop-up menu for each object, the When menu item lists the events for which you can add processing code. Figure 1 shows the list of events for an entry field in VisPro/C.

Selecting an event from the When menu opens an editor window for that event. While you can simply type code in this window, the real power of VisPro is in its drag-and-drop programming functions. Figure 2 shows part of the program for handling the When Clicked event on the ADD button. To clear the entry field, simply drag the entry field to the editor window and drop it. A dialog box listing all of the OS/2 constructs that make sense in this context pops up. One of them is Set Item Text. If you select this construct, a C or C++ statement to set the item text is placed in the window. You'll need to edit the statement to set the item text to the null string, but that is all there is to it.

In addition to the Layout view, forms also have Settings, List, and Event Tree views. You can switch views or open additional windows with other views. In the Settings view, you define general characteristics of the window, including window title text, frame control flags, timer, colors, and menu structure. The list view has a container showing all objects on the form, any text associated with the object, the symbolic name, control identifier, and help identifier. You can change any of these values by direct editing or by using the right mouse button to pop up the menu for that object.

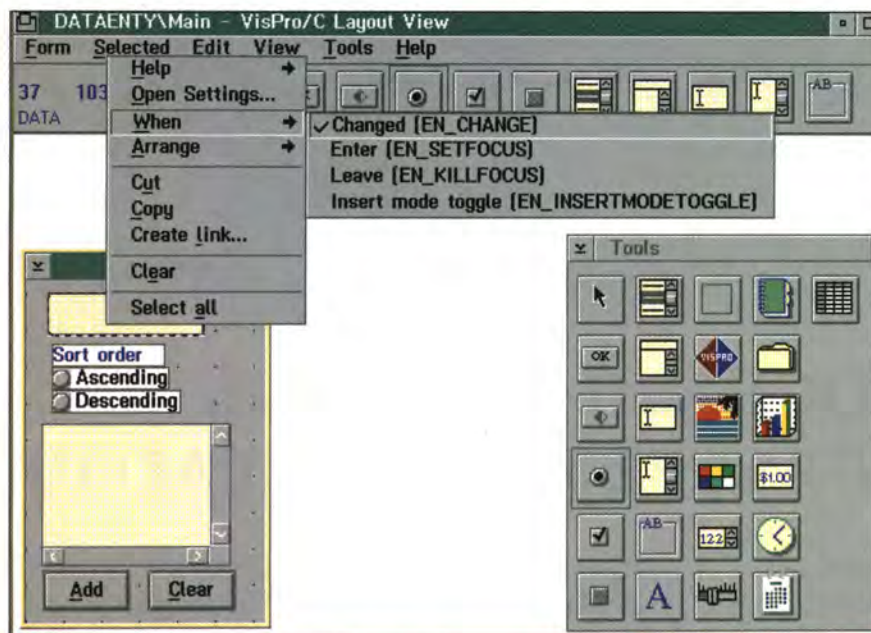


Figure 1. Events for an entry field in VisPro/C.

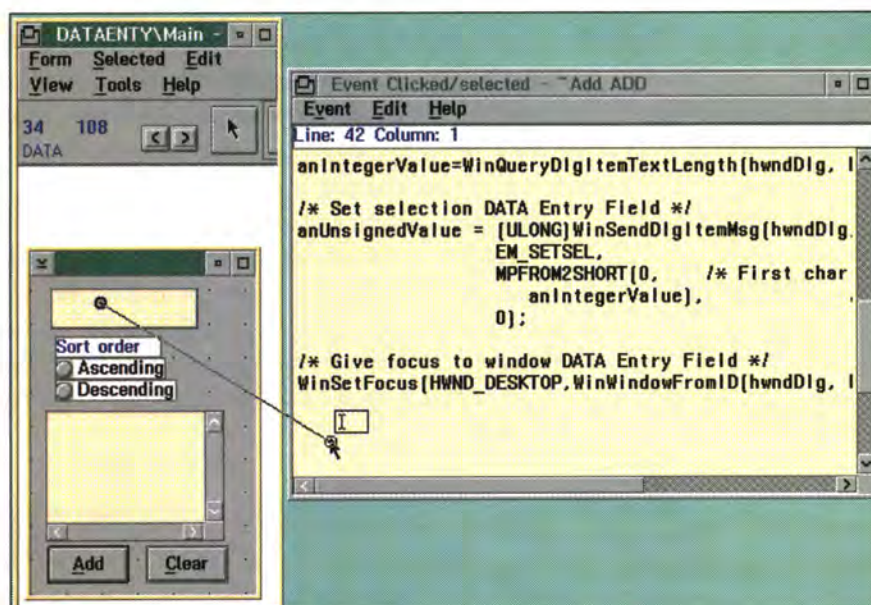


Figure 2. Clearing an entry field using drag-and-drop programming.



The Event Tree view, shown in Figure 3, offers an exceptionally powerful view of the form. All objects and all events defined for those objects are listed. Select any event, and the code for that event is shown on the right. All of the drag-and-drop programming functions work here too, as do the menus for the objects. This view is an application browser that allows you to view the application's objects, events, and processing code conveniently and easily.

The project folder also contains an Options object. Use this object to set the name of the executable file to something other than the default RUN.EXE. Set the application description and copyright notice here. Compiler, linker, and other options are in this notebook too.

When you select the Generate or Generate and Compile options for the form, VisPro places the generated application in the Source directory. You can place any additional source code required by the application in the SubProcs directory. Any code in that directory will automatically be included in the generated makefile and compiled with the application.

The Resource object defines resources to be included in the RC file: bitmaps, pointers, and strings. You can drag bitmaps and other resources to this object and drop them in, or open the Resource object and add the resources directly.

DIFFERENCES BETWEEN VISPRO/C AND VISPRO/C++

VisPro/C and VisPro/C++ have the same user interface and are used in the same way. Since VisPro/C is designed to support the Presentation Manager API and VisPro/C++ is designed to support the ICLUI class library, some differences exist between the two products. Events and code are necessarily expressed in terms of the target platform. VisPro/C supports

sizable windows and the value set control; VisPro/C++ does not in this release.

VisPro/C has a code template subsystem. You can browse through the list of code templates and drag the desired template to the VisPro/C event or code section window. Or you can copy the desired template to the clipboard and paste it into your editor. You can add your own templates to develop a drag-and-drop template system that meets your specific needs.

VisPro/C++ has an ICLUI class library browser. With this browser, you can view the entire ICLUI class library and drag a template for any class method into your code window. The browser examines the ICLUI's .HPP files, so changes to the ICLUI will be reflected automatically in the browser. You can extend the browser by adding your own class libraries.

DATABASE DESIGNER

The Database Designer is an unexpected surprise in a visual programming tool. Not only does it generate SQL statements for

DB2/2, but it is also an entity relationship diagramming (E/R) tool. If you have an existing database, the Database Designer will reverse engineer the database and create an E/R diagram for it. You can also use the Database Designer to design a new database, including referential integrity and creating foreign key relationships. After you have created the database, you can export data definition language (DDL) statements to create the database or export a DDL macro (a REXX command file) to create the database. While the Database Designer is not as sophisticated as a dedicated E/R modeling tool, it is far easier to use than DB2/2's Query Manager for defining a database.

One way I evaluate a visual programming tool is to write a small application to update a database table. With both VisPro products, the steps are easy. Start the Database Designer and select Database/Reverse Engineer. Pick the database. Create a new project and open the Main form. Drag the table from the Database Designer to the form and drop it. Select the columns and application features

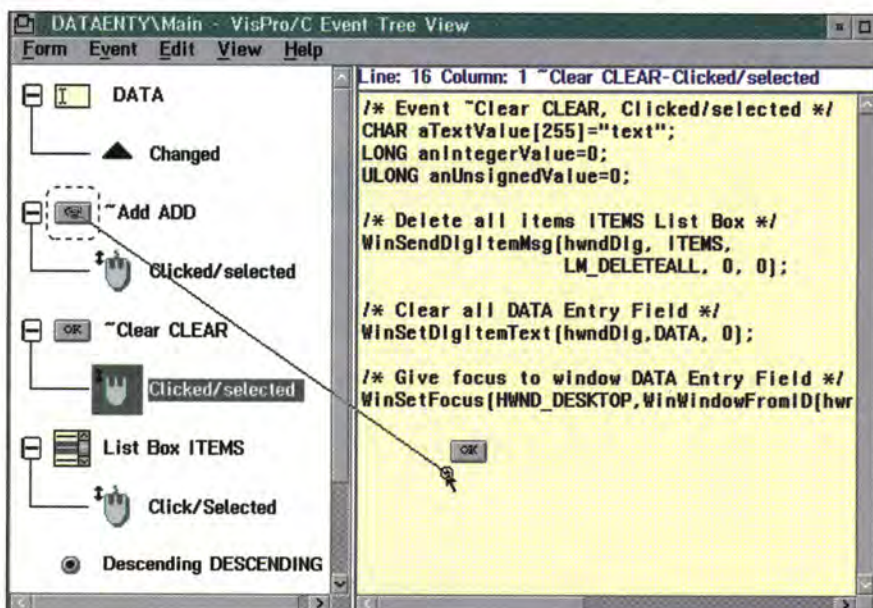


Figure 3. Event Tree view of a form.

desired. VisPro will generate the objects needed for the application and will create event processing code to implement the table maintenance functions. The generated code for the Search push button is a skeleton that reads all the entry fields but does not construct the WHERE clause on the SELECT statement. But that's easy to add. With all features enabled, some cosmetic tinkering, and modification of one `sprintf` statement to partially implement the search push button, you get an application like the one shown in Figure 4.

The Database Designer will also help you write syntactically correct SQL. Just drag the database tables to an event window in your application, and a window to create an SQL statement opens. This window will walk you through creating an SQL statement with all of the desired parameters. When the statement is complete, select OK to add it to your event window.

ADD-ON OBJECTS

Both products add graphical objects for special functions. Spreadsheet, business graphics, formatted entry field, clock, and calendar ob-

jects provide ready-made answers to these special needs. Figure 5 shows a simple application with spreadsheet, business graphics, and calendar objects.

The spreadsheet object provides a spreadsheet visual object with a simple cell-editing function. The spreadsheet notifies you when a cell has been changed. The sample program shown in Figure 5 updates the graph when a cell is changed. The spreadsheet graphical object stores any strings entered in the cells. However, there is no spreadsheet engine to interpret cell contents, implement formulas, or perform functions often required in a spreadsheet. This object is more of a table than a spreadsheet.

The business graphics object offers pie, bar, line, area, and stacked bar charts with optional three-dimensional appearance. Additional styles display a legend, horizontal scale lines, or a border. You can set these styles during the design phase and change them during execution. The graphics object is a display only object; mouse input is not supported.

The formatted entry field object supports numeric, alphanu-

meric, uppercase only, date, time, currency, and a picture format. Date, time, and currency formats reflect the National Language Support values set for the system. Methods to set the current date and time are provided.

The picture format allows you to create entry fields that allow only digits, alphabetic values below a specified character, or uppercase characters. You could use the picture clause `999-99-9999` for a social security number or `ZZZZ99-FFFF` for a string that had four uppercase letters followed by two digits, a hyphen, and then four uppercase characters each from A through F.

I found that the format options provided good output capabilities. Input and editing are more problematic, however. For some formats, such as date and picture, you are always in overtype mode, regardless of which mode you select. That will often be acceptable, but the visual appearance does not reflect that mode. Cutting and pasting a picture field included only the first character in my tests. Overall, formatted entry field support works best for input when the number of characters to be typed is preset. With all of these fields, the retrieved field value includes any punctuation and other characters.

The clock object is a running, visual clock with the same appearance as the standard OS/2 System Clock in the System Setup folder. It can be set to analog, digital, or military (24-hour digital) format. The calendar object provides an attractive calendar, as shown in Figure 5.

DLLs for each object can be distributed with your application royalty-free. Source code for the add-on objects is not included. If you want to add your own objects to VisPro to make it easy to design applications using your objects, a VisPro Objects program is available from HockWare on request.

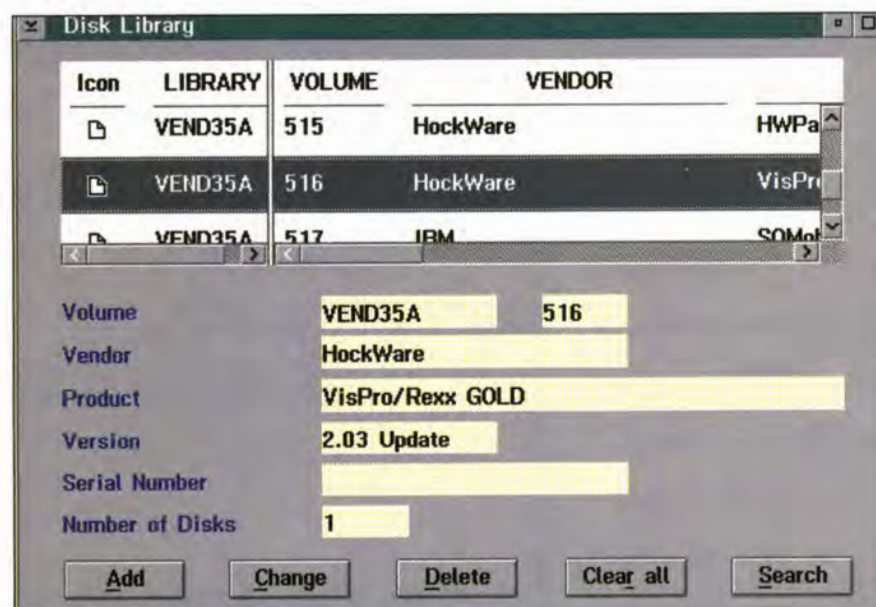


Figure 4. Simple database table query and update application.

EASE OF USE

Overall, I found VisPro/C and VisPro/C++ to be exceptionally easy to use. These tools are integrated with the Workplace Shell, resulting in a different usage style than tools that provide a self-contained development environment. For example, you open your project folder to run or debug your application rather than using menu items in the VisPro window.

Surprisingly, the form you work on is a child of the VisPro application instead of the desktop, like the tool dialog. This means you must size your VisPro application window to include the entire window you are designing. The generated code is well commented and formatted using the K&R style. All of the code you have added for events and form processing are set off by `##START` and `##END` comments, making them easy to find. VisPro/C embeds your added code directly in the window's `switch` statement, which makes for a large window procedure.

DOCUMENTATION

VisPro/C includes a 362-page manual; VisPro/C++ comes with a 323-page manual. The contents and structure of the manuals are similar, with introductory and reference sections. Each object and its styles and events are described. The introductory section includes step-by-step instructions on creating an application with a button and an entry field.

The final section of the manual walks through two sample applications in detail. The first is a simple application with radio buttons, entry field, listbox, and two push buttons. The second is a more complex example that demonstrates use of container, notebook, and spin button objects; multiple forms; and bitmap graphics.

Online documentation includes a short `README.INF` file

and help files. Most menu items and tools have help pages called up by F1. Help pages include some hyperlinks to related topics.

The editor supports KWIKINF, allowing you access to IBM's OS/2 help files, including toolkit and compiler help files. Highlight an OS/2 API, press Alt+Q, and the Presentation Manager Reference for the API is displayed.

Five sample applications are included: simple data entry, calculator, database, business graphics, and a multiwindowed Multiple Listing Service. The examples demonstrate most of the add-on objects and Presentation Manager controls.

WARTS AND BLEMISHES

While there are minor fit and finish problems typical of a version 1.0 product, the products are stable. The only serious problem I encountered was in the Database Designer; the Designer simply closed when asked to reverse engineer some databases. HockWare sent me a copy of the latest revision, and that fixed this problem for my databases. Most of the other problems I saw were nui-

sance items; VisPro never crashed in my testing.

Online help is uninspiring, and the help page presented is often not the one of most interest. For example, it would be nice to select an object in the form, press F1, and get help for that object. Instead you get general help for VisPro, so you have to go to the Contents page and select help for the object there.

HPFS naming capabilities are not fully supported. If you name your project with embedded blanks or special characters, you'll be unable to build the application.

VisPro lacks a status or information line, so it cannot display information about the object under the cursor. If you don't remember what the icon for an object looks like, you have to guess, since all of the buttons for tools are graphical. A status line could have identified the object.

Both products have a distressing tendency to use numbers instead of symbols if you don't define a symbolic value for an identifier immediately. This results in statements like `case 256:`. And you cannot easily change the number to a symbol later.

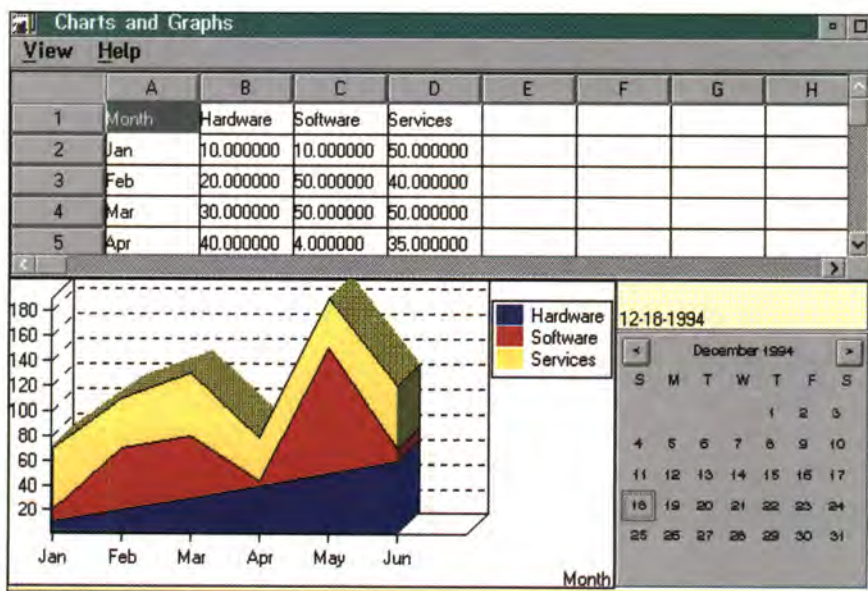


Figure 5. Spreadsheet, business graphics, and calendar objects.



LIMITS OF APPLICABILITY

There is no support for printing. You're on your own to develop printer dialogs and to do OS/2 Presentation Manager printing.

VisPro creates windows in the resource script using `DLGTEMPLATE` and opens the windows with `WinLoadDlg`. There is no support for creating a client window that you can paint in response to `WM_PAINT` messages. Both products are well suited to typical business applications and utilities where GPI graphics functions are not required.

Each product comes with a program to convert a VisPro/REXX application to VisPro/C or VisPro/C++. This conversion program makes it easier to develop a prototype in REXX and then convert the application to C or C++. The conversion program converts only the form and object layouts; the REXX source code is not converted or copied as comments, which would make recoding the events in C or C++ easier.

VisPro can be used for creating initial source code for an application that will later be maintained manually. When you begin manual maintenance, you will probably want to reorganize the code to replace any numeric identifiers in the generated source code with

symbolic names and to modularize the window procedures. If you use VisPro throughout the life of the application, the exact form of the source code should not be an issue.

TECHNICAL SUPPORT

HockWare provides technical support by fax, telephone, electronic mail, and a CompuServe forum (OS2AVEN section 8, HockWare; GO HOCKWARE to get there). While the documentation states that support is provided for only 90 days from date of purchase, I have never seen support refused on CompuServe. Users of the VisPro series of products congregate in the HockWare forum and answer questions for each other as well as get answers from HockWare staff. HockWare has been good about sending updated versions to resolve problems.

SUMMARY

VisPro/C and VisPro/C++ make writing OS/2 Presentation Manager programs in C and C++ quick and easy. Standard OS/2 controls are supported and useful custom objects are included, allowing the developer great flexibility in designing the application. The Database Designer creates a skeleton DB2/2 database application for

you and aids writing SQL statements. The drag-and-drop programming style allowed me to create applications quickly and to concentrate on the operation and intent of the application rather than on the syntax and programming details.

Guy Scharf is president of *Software Architects Inc.*, a software development firm specializing in developing OS/2 Presentation Manager applications for vendors and business. He can be reached via e-mail at 76702.557@compuserve.com or at Software Architects Inc., 2163 Jardin Drive, Mountain View, Calif. 94040.

PRODUCT INFORMATION

VisPro/C.....\$399

VisPro/C++.....\$399

VisPro Development
Suite.....\$599
(Includes VisPro/REXX, VisPro/C,
and VisPro/C++)

Vendor: HockWare Inc.
P. O. Box 336
Cary, NC 27512
Tel. (919) 380-0616
Fax. (919) 380-0757

TAKE OS/2 WARP BEYOND

THE LIMITS OF SPACE

WITH NEW STACKER 4.0

FOR OS/2 & DOS.



Once again, we've stretched the limits of data compression. Because with new Stacker® 4.0 for OS/2 & DOS, you'll now be able to store an average of 2.5 times as much data on your hard disk. And you can do it up to four times faster than with previous versions.

Stacker 4.0 is compatible with OS/2, Windows and DOS applications, and new OS/2 Warp. And if you're running DoubleSpace™, DriveSpace™ or SuperStor/DS, Stacker's conversion program is the easiest way to move to OS/2 without uncompressing your drive.

Stacker AutoSave™ helps put the squeeze on lost data by backing up and protecting vital file system

information. And the new Stacker Toolbox™ gives you instant access to your Stacker tools from your OS/2 desktop—just point and click.

Stacker 4.0 for OS/2 & DOS is now available for substantially less than previous versions and at special upgrade and cross-grade rates for current Stacker users. And we go to every extent to keep you happy with our 60-day, money-back guarantee.

Call **1-800-522-7822, ext. 8601** or contact your local dealer. Then get Stacker 4.0 for OS/2 & DOS, and give your hard drive more room to breathe.

Stac
STORAGE & COMMUNICATIONS

NOTHING EXPANDS YOUR HARD DRIVE LIKE NEW STACKER 4.0 FOR OS/2 & DOS.



Outside of the U.S. call 1-619-794-4333 or fax 1-619-794-4575 for more information.

© 1994, Stac Electronics. Stac and Stacker are registered trademarks, and Stacker Toolbox and Stacker AutoSave are trademarks of Stac. OS/2 is a registered trademark of IBM. MS-DOS, DoubleSpace, DriveSpace and Windows are trademarks of Microsoft. All other product names are trademarks of their respective owners. Novell Yes: Developer tested only. Novell makes no warranties with respect to this product. Actual compression results may vary.

Circle Reader Service Number 7



Our cat-owned explorers begin a series of discussions on printing, a topic long glossed over by many.

BY MARK BENGE and MATT SMITH

This Isn't the Gutenberg Press



Mark Benge



Matt Smith

In 1436, Johannes Gensfleisch Gutenberg invented the process of creating movable type, which made continuous text. This movable type was the technique of casting individual letters in metal. The process of using paper made by hand and a wooden printing press, where the type was set, remained unchanged until the early 1800s, when papermaking machines were invented. The first mechanical presses appeared in 1811. By this time, the technique of creating casts for whole pages was perfected. Eventually, mass production of printed documents was possible.

Knowing this occurred many, many years ago, you sometimes feel as though you were back in the middle ages because you just can't figure out how to get that #@\$% text that is in one of your windows in OS/2 out to that *%&\$ laser printer of yours. Ain't technology just wonderful?

THE PRINTED PAGE

We are about to embark on yet another journey across the OS/2 Presentation Manager landscape. This time, we set our sights on printing—something so many of us want to be able to do, yet somehow just can't quite make come together.

To begin our journey, let's first take a trip to the OS/2 Presentation Manager desktop where, more likely than not, we will encounter a printer object or two.

The printer object that appears on the desktop is used to provide the interface between the printer driver, the output port (for example, LPT1, COM1, and so on), and the printer queue. It also provides the visual interface that allows the user to interact and set up the printer.

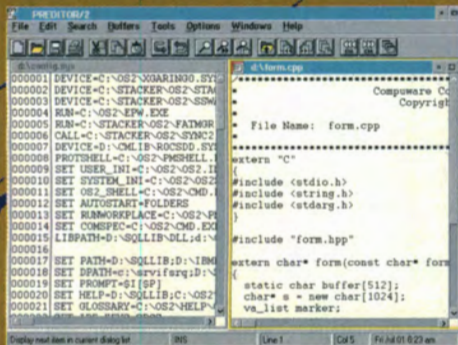
When you ask for the settings of the print object, you will be presented with the notebook pages that allow you to set up the printer. The contents of the notebook particularly interest us since we will eventually have to deal with the settings selected here when we print.

The first page of the notebook, **View**, contains the name of the printer. This name is generally used to denote the actual printer being used. It can be seen within the queue directory and is where the print jobs are spooled. You can't actually do anything with this page; it is more informational than anything.

The next page, **Printer Driver**, is used to do many things. If you have more than one printer installed on your system, they will be presented within the **Printer driver** area. The printer driver associated with the printer object selected will gen-

For the untamed
programmer
only.

PREDITOR



Introducing PREDITOR/2.

Your source code has never fallen into the grips of a more powerful editor. PREDITOR/2 is for developers who don't have the time to deal with rigid programming environments, who must change facilities to fit their exact requirements and who need to dramatically extend editing functions to attain peak productivity. With this editor, you'll work with the speed and strength of a true predator.

HIGHLIGHTS:

- CUA-compliant GUI
- Multiple document interface
- BRIEF, VI, EMACS, ISPF, CUA emulation
- WorkFrame/2 integration
- Built-in compiler support
- Extensive customization facilities
- Powerful C-like extension language
- Unlimited Undo and Redo
- Powerful search and replace functions
- No limits on file sizes, buffers or windows

Special Introductory Price... \$149.00 • MSRP \$249.00

To order PREDITOR/2 or to receive a brochure, call now. 1-800-535-8707 or fax: 1-810-737-7119

PREDITOR/2 is a trademark of Compuware Corporation. All other company or product names are trademarks of their respective owners.
© 1994 Compuware Corporation.

Circle Reader Service Number 8



erally be highlighted. You select the printer driver to use for the current object from those presented.

The area below the printer driver list, **Default Printer Driver**, will contain the printer driver selected for the object. This should not be misconstrued as the default printer driver for the system—that is set through the object pop-up menu **Set default item**.

If you want to change the default driver for the printer object, you only need to select one of the drivers in the upper area, and the selected default driver in the lower area will be replaced by the new one.

The way you set up the printer driver object within this **Printer Driver** page is by either double-clicking the mouse pointer on the object or clicking the right mouse button on the object and then selecting the **Settings** menu item, as shown in Figure 1.

This causes the **Printer Properties** dialogue to be displayed so you can assign trays to a form, define a new form (for exam-

ple, new paper size), and set the download fonts and other device defaults such as duplex and resolution. The values selected will apply to all devices handled through the output queue for the device. This means that when you have one printer object physically linked to two printers, both printers will use the properties.

Unlike the **Job Properties** dialogue, which tends to be applied only for a print job, they remain persistent—a gazillion-dollar, object-oriented buzzword meaning they stay set. We will not cover the **Job Properties** dialogue in this issue.

The **Output** notebook page is used to select the location where the print object is to output. Some interesting things should be noted here. It is entirely possible that if you have two identical printers attached to your system, you can have the output for the printer object printed on both printers simultaneously. This is done by simply selecting the output ports that the printers are attached to. For this to work properly, you want to make

sure that the printers are similarly set up. Otherwise, if you have an HP 4 and a Lexmark 4039 Plus in different print languages, the output for one of the printers will probably be correct with the other likely to appear as gibberish.

We should probably explain some of the visuals since they are not well documented and have somewhat different meanings than what you might be used to. A solid background behind the port image indicates the printer object is using this port for output. A heavy-hatched background (diagonal lines) indicates the port is being used by the selected printer object. A light-hatched background indicates the port is being used by another printer object. No background indicates that the port is not being used by any printer objects. When you want to output to a file, you only need to select the **Output to file** check box. This will cause the selected port to be deselected.

The **Queue Options** notebook page is used to select the queue processor that is to handle the printing process. In essence, it acts like an air-traffic controller. It ensures that print jobs are properly sequenced for access to the print object, just like airplanes waiting to land at the airport.

Usually, you will see only one of the print queue processors, **PM-PRINT**, within this page. The second queue processor, **PM-PLLOT**, will appear only when you have a plotter object installed on your system.

A little explanation regarding the options is in order here. The first option, **Job dialogue before print**, is used to force the **Job Properties** dialogue to be displayed before each and every print job. The **Printer-specific format option**, when selected, will cause the output to be in "raw" mode (which in plain English means a format that

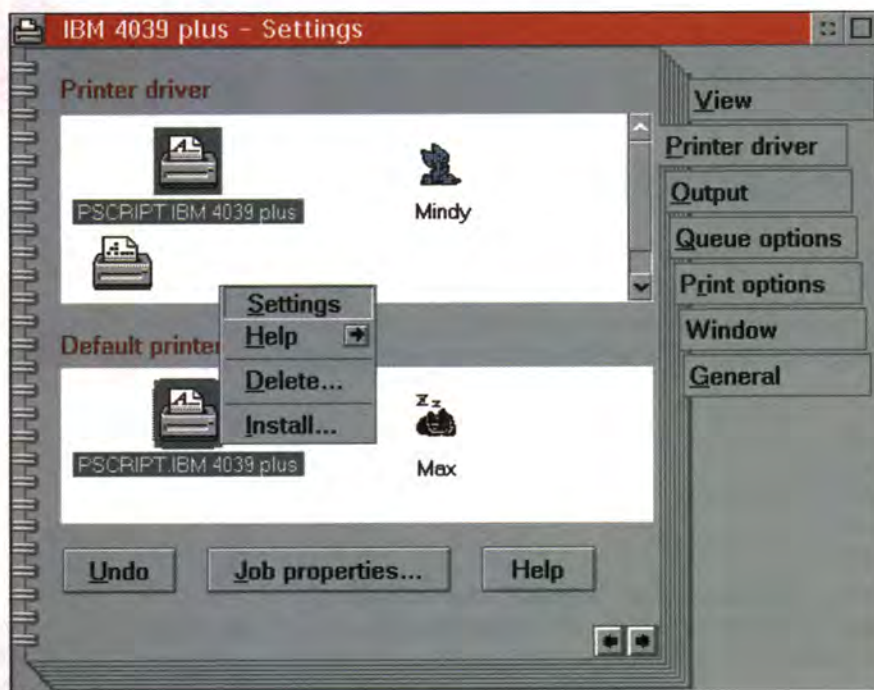


Figure 1. Print Object settings page.

is in the printer's native language). This will override any selections made programmatically. It should be used as a possible workaround only when something doesn't work.

We should also note that this is the format used by all non-Presentation Manager applications. Remember all those wonderful printer drivers you had to wade through in the various word processing programs? Well, they had the delightful job of creating this kind of output, but that is a separate topic we don't want to touch.

The last option, **Print while spooling**, simply allows the multitasking nature of OS/2 to be used while you print. This allows the job to start printing immediately instead of waiting for the entire job to be spooled before any actual output hits the printer. This is recommended for Presentation Manager applications.

The other notebook pages do not apply to our discussion so we will ignore them.

Let's go back to the printer object as an icon on the desktop. When you double-click on the printer object, you are presented with a window that will show the jobs that are currently active. This visual interface is provided courtesy of the queue.

UNDER THE HOOD

With this newfound understanding of the high-level viewpoint of the printing process, let's look at the parts that make the printing process work under the hood. Some of the components are contained within the `OS2SYS.INI` file as strings.

Back in the good ol' days of OS/2 1.x, you actually had to muck around with these strings to determine what printers were set up on a machine. OS/2 2.x and 3.x simplify the process somewhat by

providing some APIs to do all the dirty work.

What do we mean by dirty work? Well, to put it bluntly, the code for this article used to be contained in the same file and was well over 1,200 lines, and that was without the printer setup dialogue! To give you an example, you used to have to query the keys within the `OS2SYS.INI` file for `PM_SPOOLER_PRINTER`. These keys would be the list of active printers on your system. Then you would have to parse (yes, the operative word here is parse), the associated string for each key. Yucko! The following depicts the arrangement:

```
PM_SPOOLER_PRINTER
IBM4039p
LPT1;PSCRIPT.IBM 4039 plus;IBM4039p;;45;
Linotro1
FILE;PSCRIPT.Linotronic 300
v49_3;Linotronic;;45;
```

Someone, thank the programming gods, was enlightened enough to see that this was a mess and created APIs within the system that greatly simplified the task of grabbing the device information from the `OS2SYS.INI` file. Through the OS/2 API `Sp1EnumQueue`, we can easily find the necessary printers that have been set up on a system.

All we have to do is record the necessary information, and we will

have cleared the first hurdle—which by all indications is the hardest. Unfortunately, it is still a hurdle since the OS/2 technical documents don't hold your hand by pulling all the necessary information together.

These APIs could be compared to the stage in the evolution of the printing press in the early 1800s. It was quite the revolution to watch all the code used for this work in OS/2 1.x simply disappear and be replaced easily by only a few lines of code.

A SIMPLE PRINT API

Although OS/2 provides a nice API to crack the printer information you need to print, you still need to record and possibly display this information to the user. One approach is to crack the printer information each time you need it. This approach has its good and bad points.

On the plus side, you use the information only when it is needed. This can save on memory, module loading, and so on. On the minus side, requesting the information can be a hit on the performance as the system gathers the information for you. On the plus side again is the fact that the printer setup will be current up to the point you request the information. On the minus side again, if

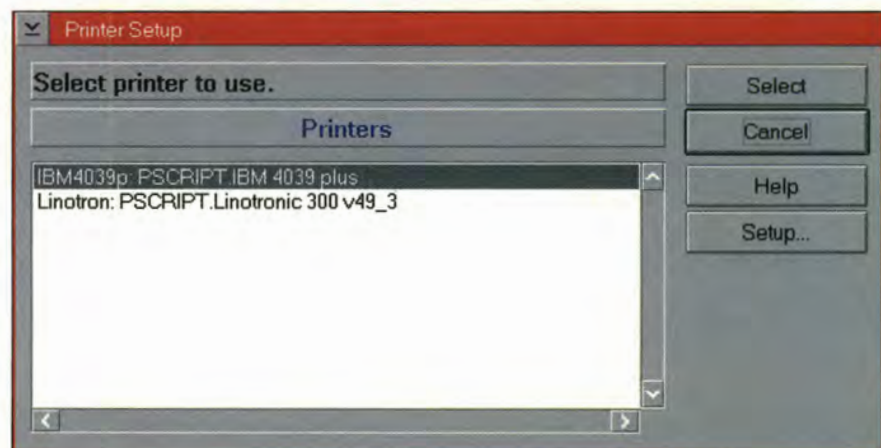


Figure 2. Printer Setup dialogue.



the user has set up the information for the printer within the current application, he or she may not want it set differently, even though he or she may have reset something on the printer itself (for example, changing from portrait to landscape).

The alternative is to query the printer information during program startup and initialization. This ensures the user has to wait only once for all the one-time waiting actions that an application may have to do.

The simple printer APIs presented here can be used as needed or during the program startup. The important issue regarding these APIs is that they are fully reentrant and do all the dirty work of caching the printer information for use later on.

BASIC PRINT INFO

The basic printer information required for successful printing includes the printer name, the print driver name, and the printer driver data. Also needed is the current default printer that has been selected by the user. This isn't found using the Spl* APIs, but through the Prf* APIs, just like the days of old with OS/2 1.x. This information is recorded under the application name of PM_SPOOLER under the key QUEUE. The string returned will be the name of the default print queue. If no queue has been defined, the first queue name provided by the Spl* API should be used.

Using the simple print API, we build the list of active printers on the system through the call PrnCreatePrinterList. This call determines the number of printers present and builds the list of print-

ers while simultaneously determining the default printer. This API allocates the necessary memory for the printer data that forms part of the list. The call PrnDestroyPrinterList is used to destroy the printer list created.

The next simple API, which we won't explain in detail until next time, is the call PrnOpenDC, which is used to open the device context for a print job. It uses the information gathered through PrnCreatePrinterList.

The PrnQueryJobProperties API is used to query the job properties for the current printer and to display the selected printers job properties dialogue.

The second to last API is simply used to retrieve a specific printer name from the list of printers. The PrnQueryPrinterName is generally used within the print dialogue to inform the user which printer is currently selected.

The final API, PrnSetupDlgProc, is used to display the typical printer setup dialogue, like that shown in Figure 2, which allows the user to select the printer from the list and to set it up in any special ways.

SIMPLE STUFF, REALLY

When you really look at it, this is not that hard. All it really requires is an understanding of the overview of the printers as objects, their parts from the Settings perspective, and the underlying constructs that are the backbone of the printing process.

UNTIL NEXT TIME

Next time, we will start the process of using this information in real ways. Stay tuned as we discuss the issue of fonts on the printer and how you can manipulate them.

Mark Benge, IBM Software Solutions, Cary, N.C., is a staff programmer who joined IBM in 1989 and has worked on various CUA '91 controls for OS/2 2.x. He works in IBM user interface class library development, where he was involved in the implementation of C++ classes for drag-and-drop in C Set++ 2.1. Mark has a B.S. in computer science from Western Carolina University. He can be reached on CompuServe at 73532,2063 and via Internet at banzai@vnet.ibm.com.

Matt Smith, Prominare Inc., Toronto, Ont., is lead architect for the Prominare Development System, an OS/2 2.x advanced GUI development environment. Matt has been actively involved with OS/2 since 1988 and cofounded Prominare in 1990. Smith has a degree in architecture from the University of Waterloo. He can be reached via Internet at msmith@interlog.com.

REFERENCES

Credits: Screen captures were done through OpenShutter from One UP Corp. and touched up through PaintBrush in Win-OS/2.

Special thanks to Larry Moore of IBM's OS/2 Printer Development Group for providing insight into the printer objects.

You can download PRNT1.EXE from these electronic sources:

CompuServe's OS2DF2 forum (OS/2 Developer Mag section).

File Area 11 on the IBM PCC BBS, (919) 517-0001.

OS/2 Tools section on the OS/2 BBS for IBM TALKLink customers.

FTP to the site address gold.interlog.com and log in as anonymous. The source is located in the /pub/prominare subdirectory.

A few words from our competitors:

Prtablty.

Prtbilty.

Portblty.

Portabit.

No matter how they misspell it, it's still not portability. At Zinc, we understand that porting the last 20% of your code takes 80% of your time.

Only Zinc offers complete portability.

With Zinc® Application Framework™ you can develop on the platform you prefer. And since Zinc is the only one that delivers 100% portability, you'll have your application on other platforms as fast as you can recompile. It's part of what makes Zinc the most productive—and affordable—tool a programmer can use.

Productivity that leads to opportunity.

Portability is just one road you'll find a little easier. Zinc zips through tedious tasks with C++ object orientation and a unique visual development tool. Globalizing your application is as easy as translating the text. And Zinc is the only product that supplies 100% of the source code.

It all adds up to productivity. Which means more profitability. Which is a concept we probably don't have to spell out for you. For free information and demonstration software, call toll-free:

800 638 8665

Outside the U.S. call: +1 801 785 8900. In Europe call: +44 (0)181 855 9918. In Asia: +81 (052) 733 4301. Contact Zinc electronically at info@zinc.com or GO ZINC on CompuServe.



z i n c

NO LIMITS



Designing and writing single-source applications for OS/2 Intel and PowerPC platforms.

By DAVID REICH

Power Applications



David Reich

As I write this, the first OS/2 Warp for the PowerPC beta is making its way to testers and application developers. It's about time to begin seeing OS/2 Warp for the PowerPC in this magazine for what it really is: OS/2. *OS/2 Developer* covers OS/2 application development, and OS/2 Warp for the PowerPC is simply OS/2 Warp running on machines based on the PowerPC processor.

If you were to put an Intel-based PC next to a PowerPC-based machine and run OS/2 Warp on both, users would not be able to tell the difference. OS/2 Warp for the PowerPC is just OS/2 with the underpinnings replaced. Rather than a monolithic kernel, with file system, scheduler, device drivers, and so on built into it, OS/2 Warp for the PowerPC is based on the IBM Microkernel, with cross-system services that live outside of the kernel and run in user mode. This is a big departure from what you have seen in the OS/2 Intel architecture, but it affects only those writing the system or system extensions, such as new cross-system services. For applications, there is not much difference in writing for the PowerPC version of OS/2 Warp.

RUNNING DOS AND WINDOWS PROGRAMS

DOS and Windows programs under OS/2 Warp for the PowerPC run in a

similar fashion to their Intel counterpart. The cross-system service, called MVM, is responsible for running your DOS and Windows applications on PowerPC-based OS/2 systems unchanged. The MVM server emulates DOS and runs the DOS and Windows programs you have today, preserving investments in existing software.

OS/2 PROGRAMS

The PowerPC instruction set is RISC and is vastly different from the Intel instruction set. While OS/2 Warp for the PowerPC emulates and runs Intel instructions for DOS and Windows applications, native OS/2 applications will not be run as compiled for the Intel platform. The good thing is that as long as you write your applications well, a port of a native OS/2 application to PowerPC is not much more than a recompile.

OS/2 PERSONALITY/API

In the Microkernel-based architecture now being used for OS/2 Warp for the PowerPC, the system is divided into layers, with clean interfaces between each. The layer that applications talk to is called the OS/2 Personality. MVM, as mentioned above, is also a personality, and it provides the DOS and Windows system services to run DOS and Windows programs as well as the Intel instructions needed to run these pro-

Migrate your code to 32-bit OS/2 NOW!



Delivers



SMART®

Source Migration Analysis Reporting Toolset

*SMART Tool Helps
Migrate Your Code*

*Get a jump on
your competition
... Get SMART!*

*Get One Up Corporation's
SMART Toolset through
The Developer Connection
for OS/2*

Now you can get SMART through The Developer Connection for OS/2.

SMART analyzes YOUR Windows™ code (16-bit or 32-bit), and OS/2® code (16-bit), sizes the conversion effort, and automatically converts the MAJORITY of YOUR code to 32-bit OS/2.

When you have completed your migration, you'll have a state-of-the-art application that exploits all the advantages of 32-bit OS/2, the leading-edge PC operating system available TODAY!

The Developer Connection for OS/2 is a CD-ROM containing lots of great tools for your OS/2 programming. It is updated quarterly, and comes with a newsletter.

How do I get SMART?

To get the complete SMART Toolset, as well as future versions of the tool, simply join The Developer Connection for OS/2. For more information or to subscribe to The Developer Connection, call 1-800-633-8266 (in Canada 1-800-561-5293).

SMART - Source Migration Analysis Reporting Toolset is a registered trademark of One Up Corporation.

®IBM and OS/2 are registered trademarks of International Business Machines Corporation. ™Windows is a trademark of Microsoft Corporation.





grams. The OS/2 personality is the dominant one, providing the user interface in the form of the Workplace Shell. The personality also provides the API and services for the applications to interface with the core system services.

The OS/2 personality consists of a set of client libraries and the OS/2 Server. These two entities combined provide the OS/2 APIs that you see in the Intel version of OS/2 Warp and will afford your applications all of the same functions on PowerPC machines that they see on OS/2 Warp for Intel. Of course other system services are callable by applications but, as you will soon see, you are best off sticking with the high-level OS/2 APIs and high-level language functions to keep your applications source-compatible between architectures.

The architectural differences between OS/2 Warp on Intel machines and OS/2 Warp for the PowerPC are not the reason for the recompilation. The chip architectures (RISC vs. CISC, along with their unique instruction sets) require recompilation. In fact, OS/2's microkernel-based architecture makes it easy to move applications from one platform to the other, and to other hardware platforms as well.

MOVING BACK AND FORTH

As you've just seen, the OS/2 Personality provides the application interface to all system services. This interface is the same OS/2 API you have been (or will be) writing to on your familiar Intel system. If you design your applications well—taking into account threads and virtual memory—and avoid coding in Intel-specific dependencies or timings, you will find that porting to PowerPC is straightforward. It means recompiling your code using a different set of tools to generate PowerPC binaries.

The tools you will be using include a compiler to generate a PowerPC binary and a set of libraries specific to OS/2 Warp for the PowerPC. The libraries with which you will be linking are specific to the OS/2 personality, as opposed to OS2386.LIB, which you would use for developing OS/2 applications for Intel. The libraries resolve the API calls you make to the OS/2 server or to the OS/2 personality client libraries that provide system services. They do this either by themselves or by making requests of the IBM Microkernel or the shared system services, such as the file system or device drivers. Note that your applications do not need to call or even know how to call microkernel services. The OS/2 server and client libraries handle all of this for you.

You will probably have to change your makefiles somewhat to account for the new compiler, specific switches for it and for the PowerPC, the libraries, and so on for the link phases. However, you should have to make little or no changes to your application source code to create a PowerPC application. The compiler you use simply generates PowerPC opcodes, as opposed to, say, C/Set++ for Intel, which generates Intel opcodes. The link step links the appropriate libraries, and voilà—you have an OS/2 application for the PowerPC.

The key to making this transition as painless as possible is to keep a single source code base, written in high-level languages such as C, and using the published API for all system services. This will also make it as maintainable as possible, which translates into cheaper development costs to have applications on multiple hardware platforms. By sticking to this, you will get high-performing, portable applications that you can leverage

across hardware platforms and that will run on whatever OS/2 your users happen to be using, on whatever hardware platform best suits their needs.

So, how do you write a good PowerPC application? Write a good OS/2 application. That's it. Experience with application developers during the early stages of OS/2 Warp for the PowerPC development, as well as throughout the beta test period, shows that OS/2 code can simply be ported and recompiled. This means writing to the system APIs and avoiding assembler language routines and all coding that circumvents the system APIs.

If you follow the easy route and use common sense, you will reap all the benefits of multiple markets for your applications without having to recode and spend more money on development and testing. As I have told developers for years, be a lazy programmer. Let the system do the work for you. It is up to the system to work the same way across multiple versions and, now with PowerPC, multiple hardware platforms. It is up to IBM to make the system services look and behave the same with respect to how application programs see and use them. As OS/2 is moved to different hardware platforms, you will see this benefit scale your efficiency and productivity to even higher levels.

David Reich has been with the IBM OS/2 development team since 1987. He has worked on many parts of the system, supported customers and application developers, and traveled the world giving seminars and teaching OS/2. He has just completed work on Designing High-Powered OS/2 Warp Applications, available this spring and published by John Wiley & Sons. He can be reached on CompuServe at 76711,632 or via Internet at speedracer@vnet.ibm.com.

Take the
"black holes"
out of
your test
coverage.

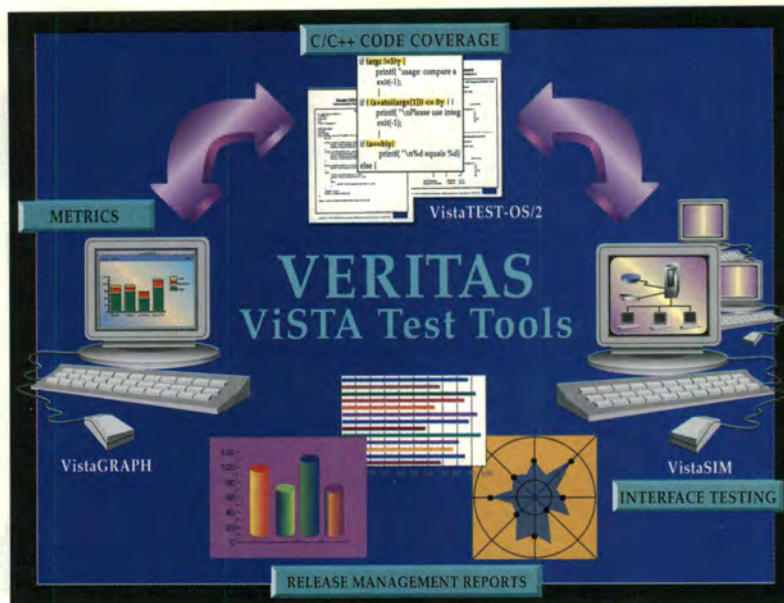


ViSTA™

**Before Your
Clients Start
Disappearing...**

Black holes may be an interesting phenomenon in outer space, but not in your software testing. We have helped 100's of companies like yours, on over 200 million lines of code, verify an amazing fact: less than 40% of "production level" code is being tested. Now that is a major "black hole".

**... Into the
Unknown.**



A POWERFUL SET OF ENTERPRISE TESTING TOOLS

VistaTEST-OS/2

- C/C++
- Multi platform
- Unit level testing
- Complexity Metrics
- Call Coverage, Multiconditional, Branch, Path, Function, Interface
- DDL/Multithreaded 16bit/32bit support

Code Coverage

VistaSIM

- Client/Server Testing
- Reuse Testing
- Automate Error Handling Test
- Over 1200 OS/2 Sim stubs for OS calls, PM calls, ANSI

Interface Testing

VistaGRAPH

- Metric summary
- Flexible Spreadsheet (diagrams and graphs)
- Quantifies testing process
- Project team oriented
- ISO 9000

Release Mgmt.

**UNIX POWER TOOLS
NOW ON OS/2**

IBM	HP	INTEL
- AIX TEAM	ADP	DIGITAL
- OS/2 TEAM	APPLE	UNISYS
- WPOS TEAM	ARBOR	CITIBANK
MOTOROLA	OSF	MEAD
NOVELL	SUN	DATA CARD

ViSTA tools help increase the quality of your application, isolate dead code and improve your client/server testing.



**Call 1-800-258-8649 today for a
Free Evaluation.**

FAX at 408-562-4334 or e-mail vsales@veritas.com

Be sure to ask about our
OS/2 Enterprise Workgroup
for extra savings.



VERITAS ViSTA, VistaSIM, VistaGRAPH and VistaTEST are trademarks of VERITAS Software.



Eicon Technology Corp. leaves nothing to chance when migrating its communications products.

By STEVE EVANS

EICON Technology Plays It Smart

When Eicon Technology Corp. chose years ago to offer its products on the OS/2 platform, few sophisticated tools were available to assist in all phases of the porting process. So the company gave Microsoft's Windows Libraries for OS/2 (WLO) a try and then shifted to Micrografx's Mirrors for a second project when WLO was discontinued. But when the third time came to port 3270 and 5250 terminal emulators from Windows to OS/2, Eicon had a choice.

THE MIGRATION OPTIONS

The options came down to using Mirrors again or a toolset called SMART—Source Migration Analysis Reporting Toolset—developed by Dallas-based One Up Corp. Mirrors and WLO are run-time tools. Basically, the resources are translated from Windows to OS/2 during development, but the source code remains Windows API calls. The source code is then relinked with the Mirrors library, and at run time, the Windows APIs call the Mirrors DLL, which in turn translates and then issues the appropriate Presentation Manager (PM) or OS/2 call(s).

Since the Mirrors toolkit was designed as a tactical solution for quickly migrating an application to OS/2, it was not intended as a long-term strategic so-

lution. SMART, on the other hand, could assist us in all phases of the migration process.

SMART consists of an analysis and reporting tool, a source migration tool, and utilities such as the resource translator. An expert system, SMART was designed to size a conversion effort and provide a road map for the migration.

We discovered three key advantages to using SMART to produce native 32-bit C code:

1. Performance is increased. The OS/2 APIs are called directly instead of the overhead of a Mirrors translation.
2. Maintenance costs are greatly reduced with a native version. With Mirrors, you have to depend on the company's support if there is a bug in its DLL. Or if a workaround is found instead, future "fixed" versions of the DLL can cause the workaround to fail at a later date.
3. A native version makes it easy to port the product to OS/2 for the PowerPC.

PREPARATION

In assembling the development team at Eicon, the biggest requirement by far was PM and OS/2 programming skills, plus we wanted OS/2 users on the team. Hiring Windows programmers presented the danger of ending up with



I chose to not have SMART insert migration notes for categories 000 and 001. For all categories, we keep the original code if it's not more than a few lines. We use SMART's flexible marking capability to add the developer's initial and the date for later reference.

Cat:000

BOOL replaced by BOOL (automatic)

Cat:010

```
// HDC      hdc      = GetDC( hWnd );           SM$:0
HPS      hps      = WinGetPS(hWnd);           // P$DR:12/13/94
```

Cat:020

original code:

```
if ( IsChild( MainDesk, ahWnd ) )
{
    hWnd = ahWnd;
```

after migration:

```
// if ( IsChild( MainDesk, ahWnd ) )           SM$:0
// {                                           SM$:0
//     hWnd = ahWnd;                           SM$:0
if ( IsChild( MainDesk, ahWnd ) )           // SM$:M2
{                                           // SM$:M2
    hWnd = ahWnd;                           // SM$:M2
    WinIsChild (,)                          // SM$:M2
//                                           SM$:C
// IsChild - Cat:020 Type:010 Area:290        SM$:K
// Replace with WinIsChild                    SM$:S
//                                           SM$:C
// BOOL APIENTRY WinIsChild(HWND hwndChild, HWND hwndParent); SM$:P
```

ported code:

```
// if ( IsChild( MainDesk, ahWnd ) )           SM$:0

// {                                           SM$:0
//     hWnd = ahWnd;                           SM$:0

if ( WinIsChild( ahWnd, MainDesk))           // P$TS:11/25/94
{                                           // SM$:M2
    hWnd = ahWnd;                           // SM$:M2
```

Figure 1. Code examples of each category (continued on page 28).

Cat:030

original code:

```
SetViewportOrg( memdc, -orglx, -orgly );
```

after migration:

```
// SetViewportOrg( memdc, -orglx, -orgly );           SM$:0
SetViewportOrg( memdc, -orglx, -orgly );             // SM$:M3
GpiQueryPageViewport (,);                           // SM$:M3
GpiSetPageViewport (,);                             // SM$:M3
//                                                    SM$:C
// SetViewportOrg - Cat:030 Type:010 Area:440         SM$:K
// Replace with GpiSetPageViewport                   SM$:S
//                                                    SM$:C
// BOOL APIENTRY GpiSetPageViewport(HPS hps,         SM$:P
// PRECTL prclViewport);                             SM$:P
// BOOL APIENTRY GpiQueryPageViewport(HPS hps,       SM$:P
// PRECTL prclViewport);                             SM$:P
//                                                    // SM$:C
```

ported code:

```
// SetViewportOrg( memdc, -orglx, -orgly );           SM$:0
GpiQueryPageViewport ( memps, &r );                 // P$DR:12/13/94
r.xRight = (r.xRight - r.yLeft) - orglx;            // P$DR:12/13/94
r.xLeft = -orglx;                                   // P$DR:12/13/94

r.yTop = (r.yTop - r.yBottom) + orgly;              // P$DR:12/13/94
r.yBottom = orgly;                                  // P$DR:12/13/94
GpiSetPageViewport( hps, &r );                      // P$DR:12/13/94
```

Cat:040

original code:

```
hdcmeta = CreateMetaFile( NULL );
```

after migration:

```
// hdcmeta = CreateMetaFile( NULL );                 SM$:0
hdcmeta = CreateMetaFile( NULL );                   // SM$:M4
DevOpenDC (, OD_METAFILE,,,);                      // SM$:M4
GpiCreatePS (,,,);                                  // SM$:M4
//                                                    SM$:C
// CreateMetaFile - Cat:040 Type:010 Area:540        SM$:K

// Replace with DevOpenDC                            SM$:S
//                                                    SM$:C
// HDC APIENTRY DevOpenDC(HAB hab, LONG lType, PSZ pszToken, SM$:P
// LONG lCount, PDEVOPENDATA pdopData, HDC hdcComp); SM$:P
// HPS APIENTRY GpiCreatePS(HAB hab, HDC hdc, PSIZEL pszlSize SM$:P
```

Figure 1. Code examples of each category (continued on page 29).

a product that looks like a Windows program.

In addition to each developer having his or her own PC, I put secondary PCs running the Windows development environment and the terminal emulator's Windows source code on individual PC carts so they could be easily accessible to each developer.

SOURCE CODE PREPARATION.

Our source code was about 200,000 lines of C code. We used PVCS Version Control Manager from Intersolv for source control mainly because it has been used at Eicon, and others within the company already had experience with it.

On the file server, I made four separate directories:

- The original Windows source code, for reference only. Access is read-only.
- The source code. Source code is processed through a beautifier utility to make maintenance easier thereafter. The source code is then processed through the SMART migration tool. Access is read-only.
- The working directory. Here a decision had to be made whether to leave the original Windows source (commented out by SMART) and SMART comments or to remove all unnecessary code. We decided on the latter to reduce clutter and make the native code as clean as possible. When the source files can be compiled without error, they are moved to the fourth and final directory.
- As mentioned before, this code can be compiled, but overall the product has not been tested, so debugging and all future maintenance is done here.

SMART BASICS

SMART has five levels of categories of difficulty for Windows APIs converted to OS/2 APIs. Categories 000, 010, and 020 are relatively au-

tomatic code replacement. Category 030 requires a change with more or less APIs (which SMART provides). Category 040 requires logic changes—similar functionality exists in the target platform, but the logic required to emulate the functionality must be reworked (for example, `CreatePatternBrush`). Category 050 is made up of Windows APIs that have no corresponding functionality in OS/2.

The analysis portion of SMART gives an effort level by placing a weighted value on each API to be migrated according to the category. We have found that an effort level of 1 corresponds to one hour of total work. (The effort level will differ with every programmer depending on level of expertise and productivity.)

The migration information is kept in a file known as a migration table. SMART currently provides three conversion tables, one for 16-bit Win 3.1 to 32-bit OS/2, one for 16-bit OS/2 to 32-bit OS/2, and another for 32-bit Win32 to 32-bit OS/2. A very helpful feature allows developers to additionally create their own customized tables.

If you find a conversion that the SMART table missed, you can add it in your own table. This customized table can also be used for later porting efforts.

SMART ISSUES

There are very few parts of the porting process SMART does not perform:

Online help conversion. SMART provides no mechanism for porting the online help from Windows to OS/2. One solution would have been to use the Mirrors help converter, which takes as input HPJ and RTF files in Windows help tag language and produces an IPF file (OS/2 IPFC tag language format). The IPF file is then compiled using the IPFC compiler, producing a HLP file.

```
// ULONG fLOptions);                                SM$:P
// BOOL APIENTRY DevQueryCaps(HDC hdc, LONG lStart,    SM$:P
// LONG lCount, PLONG aIArray);                      SM$:P
// BOOL APIENTRY GpiSetDrawControl(HPS hps, LONG lControl, SM$:P
// LONG lValue);                                      SM$:P
// BOOL APIENTRY GpiResetBoundaryData(HPS hps);        SM$:P

ported code:

// hdcmeta = CreateMetaFile( NULL );                  SM$:0

memset (&DevOpenStr, 0, sizeof (DEVOPENSTRUCT));      // P$TS:12/01/94
DevOpenStr.pszDriverName = "DISPLAY";                  // P$TS:12/01/94

hdcmeta = DevOpenDC (hab,                               // P$TS:12/01/94
                    OD_METAFILE,                        // P$TS:12/01/94
                    "*",                                // P$TS:12/01/94
                    9,                                  // P$TS:12/01/94
                    (PVOID) &DevOpenStr,                // P$TS:12/01/94
                    (HDC) 0);                            // P$TS:12/01/94
if (hdcmeta)                                           // P$TS:12/01/94
{                                                       // P$TS:12/01/94
    SIZEL SizeI;                                       // P$TS:12/01/94
    DevQueryCaps (hdcmeta,                             // P$TS:12/01/94
                  CAPS_WIDTH,                          // P$TS:12/01/94
                  2,                                    // P$TS:12/01/94
                  (PLONG) &SizeI);                     // P$TS:12/01/94
    hpsmeta = GpiCreatePS (hab,                         // P$TS:12/01/94
                           hdcmeta,                    // P$TS:12/01/94
                           &SizeI,                     // P$TS:12/01/94
                           PU_PELS |                    // P$TS:12/01/94
                           GPIF_DEFAULT |               // P$TS:12/01/94
                           GPIT_NORMAL |                // P$TS:12/01/94
                           GPIA_ASSOC);                 // P$TS:12/01/94
    GpiSetDrawControl (hpsmeta, DCTL_BOUNDARY, DCTL_ON); // P$TS:12/01/94

    GpiResetBoundaryData (hpsmeta);                     // P$TS:12/01/94
}                                                         // P$TS:12/01/94

Cat:050

/* cascade the windows*/
// Forward_WM_MDICASCADE(MainDesk, 0, SendMessage);
FORWARD_WM_MDICASCADE (MainDesk, 0, WinSendMsg (, (ULONG) ,
(MPARAM) , (MPARAM) ) );
//
// FORWARD_WM_MDICASCADE - Cat:050 Type:030 Area:810
// There is no native MDI support in OS/2
```

Figure 1. Code examples of each category (continued from page 28).

I decided not to do this because the Windows help system is more primitive than OS/2's, and we would end up with not only a restricted help system but one that looks like a Windows help file. The goal of the online help is to have context-sensitive help for each entry field in the program. We instead adopted a help-writing tool from IBM called Hyperwise. It basically creates help from the ground up; however, the technical content of the Windows help can be reused.

Additional manual conversion for the help system included creating help tables and subtables in a resource file and implementing a message handler in the source code; here the source code is linked to the help table and subtables.

Font conversion. Access for OS/2 provides custom-made fonts as FON files, which are in the OS/2 font file format. SMART does not provide a font converter. We used a font converter from the old Microsoft WLO porting tool; it takes as input Windows FNT files

(font source code) and produces FNT files in OS/2 format. Then we produced a font file as described in the PM Programmer's Guide. Basically, a DLL is generated from an empty C source file and the FNT files, then the DLL is renamed to a FON file.

DESIGN ISSUES

A goal in product functionality was to duplicate the feature in Access for Windows; another was to enhance Access by taking advantage of OS/2-specific features. The following lists some of the changes in design:

Installation program. We wanted it to have an OS/2 look and feel. For its OS/2 products, Eicon has standardized on Software Installer from IBM to create its installation programs. Highlights of the functionality are CID capability and uninstall capability for this version and previous versions of Access for OS/2.

MDI. The MDI interface from Windows does not exist in OS/2. Access for Windows implements 5250 and 3270 display and printer sessions as MDI client windows. We didn't implement the same functionality; the MDI code was stripped out of the source, and session windows were implemented as child windows. A good sample program that implements MDI-type functionality can be found on the companion disk of the book *Real-World Programming for OS/2 2.1*.

Keyboard handling. The message handling for keyboard input in OS/2 has a totally different structure than in Windows. SMART does not migrate this very elegantly. One developer volunteered to be the keyboard expert, and he virtually implemented keyboard handling from scratch.

REXX support. Access for Windows provides Visual Basic support for its scripting languages. The OS/2 version instead provides

The following is a simple example of an entry from our customized table.

```
.Keyword =010 010 990 SendMessage
.Command =
    =SmMigrateKeyword ("WinSendMsg ($P1, (ULONG) $P2, (MPARAM) $P3,
(MPARAM) $P4)")
```

You can also do conditional tests on parameters and use if-else constructs for branching:

```
.Keyword =010 010 990 InvalidateRect
.Command =
    =if $P3 = "TRUE" then
    =do
    = SmMigrateKeyword ("WinInvalidateRect ($P1, $P2, FALSE)")
    =end
    =else
    =do
    = if $P3 = "1" then
    = do
    = SmMigrateKeyword ("WinInvalidateRect ($P1, $P2, FALSE)")
    = end
    = else
    = do

    = SmMigrateKeyword ("WinInvalidateRect ($P1, $P2, FALSE)")
    = Note = "MANUAL MIGRATION: For " $KEY " with the fErase flag
set to " $P3
    = SmOutputNote (Note)
    = Note = "If this flag is FALSE, you must track this state and "
    = SmOutputNote (Note)
    = Note = "return FALSE when you process the WM_ERASEBACKGROUND
message. "
    = SmOutputNote (Note)
    = end
    =end
```

Figure 2. Customized conversion table entry.

REXX support; there isn't much work involved in "REXXizing" the source code to do this.

Drag-and-drop support. Files that are used to store information that is produced by Access (such as communications configuration, keyboard configuration, and macro files) can be created or modified by dragging and dropping the file on that module's icon.

CONCLUSION AND CAVEATS

Some development hints:

- If possible, start the project by migrating small modules to get the hang of using SMART.
- Try to group the source code

between the developers according to subject.

- Use an incremental, circular development approach. When porting an executable module, strip out as much functionality as possible with #ifdefs to get at least a shell up as soon as possible. Add each feature until the functionality of the Windows product is reproduced. Then add OS/2-specific features if applicable.

Steve Evans is a consultant specializing in OS/2 application development, particularly in migrating Windows applications to OS/2. He received a B.S. in computer science and mathematics from McGill

University, Montreal, Que. Eicon Technology Corp. is a worldwide provider of advanced internet-working solutions for branch office connectivity. The company is based in Montreal.



REFERENCES

SMART Source Migration Developer's Guide (One Up Corp.).

OS/2 2.0 Programming Guide Volume III (IBM Doc. 10G6495).

Microsoft Windows 3.1 Programmer's Reference (Microsoft Press).

The Award Winning, Multi-Function Enhancements for OS/2®

NEW
Version 1.51
with
Warp support!

DeskMan/2™

DeskMan/2™ Workplace Shell™ Manager

Efficiently Manage Your Workplace Shell!

- Easy to use drag & drop interface
- Complete command line interface + REXX & CID support
- Manage standardized or personalized desktops, customizing access to features
- Selectively save and recreate objects; even transport them between desktops, computers & versions of OS/2
- Recover from or prevent accidental changes to your desktop
- Much more - total management of the Workplace Shell

VUEMan/2™ Virtual Desktop Manager

- Makes multitasking easy; use and organize many concurrent activities with an easy to use drag & drop, point & click user interface
- Switch effortlessly between as many as 81 "virtual desktops" - like having 81 monitors that fit right on your desk!
- Drag and drop objects from WPS to virtual desktops, and between virtual desktops
- "Layouts" record window position and size for automatic repositioning and re-sizing at any future time
- Password protection for windows

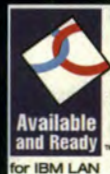


All this functionality & much more Four Great Solutions - One Low Price!

"DeskMan/2 dramatically improves the ability of corporations and users to get the most out of OS/2. DeskMan/2 helps administrators restrict and control what users are doing and provides robust backup/restore features and seamlessly enhances the features of the WPS."

Dr. Mike Kogan:

Lead Architect of OS/2 2.0
Author: The Design of OS/2



DM2Image™ Configuration Snapshot Facility

Saves Your Working Environment!

- Save or restore an unlimited number of complete system configurations (including active or inactive WPS desktops)
- Terrific for disaster recovery; training & demo machines; other uses
- Extensible rules let you customize configuration definitions
- Configuration snapshots are compressed for maximum efficiency
- Protects WPS desktop, config.sys, startup.cmd, .INI files, autoexec.bat, WIN-OS/2 desktop & system config. data

DeskMan/2 Extensions Workplace Shell Extensions

Configure the Workplace Shell to Work Your Way!

- Modify and enhance the runtime behavior of the Workplace Shell
- Enjoy drag & drop without using special keys to shadow or copy
- Control, selectively or globally, all WPS defined object styles, menu items and capabilities - the ability to Arrange, Sort, Delete, Rename, Settings, Lockup, Shutdown, etc. (even object visibility!) is completely definable
- Corporate Edition available with password protection for objects, audit trailing for WPS activities, and other custom enhancements

Copyright 1994 Development Technologies, Inc. All rights reserved. DeskMan, DeskMan/2, VUEMan, VUEMan/2 and DM2Image are trademarks of Development Technologies, Inc. CompuServe is a registered trademark of CompuServe, Inc. Workplace Shell is a trademark of International Business Machines Corporation. IBM, OS/2 and TalkLink are registered trademarks of International Business Machines Corporation.

DevTech

Development Technologies, Inc.
Software Development & Technology Transfer
308 Springwood Road, Forest Acres, SC 29206-2113
Voice: (803) 790-9230 Fax: (803) 738-0218

A Special Report from the publishers of OS/2 Developer!

Quantities are limited.
Order now!
For fastest service,
fax your order to us
at 415-905-4967.



Smalltalk



☒ **YES!** Please send me _____ copies of Smalltalk!

Each copy is only \$9.95 in the US or \$12.95 in Canada.

Please add \$2.50 for shipping and handling.

- ☐ My check is enclosed
- ☐ Charge my credit card: ☐ Visa ☐ MasterCard ☐ American Express

Name _____

Signature _____

Date _____

Send them to:

Name _____

Address _____

City _____

State _____

Postal Code _____

Mail: Smalltalk
P. O. Box 7046
San Francisco
CA 94120

Phone: 1-800-444-4881

Fax: (415) 905-4967



*As the software world moves inexorably toward the future, are you sitting on a 16-bit time bomb? Converting a large 16-bit application to the 32-bit model can be a very nontrivial task (probably why you have not done it already), but hopefully we can help you avoid the major pitfalls. **By DEAN RODDEY***

Taking the Plunge: Converting an OS/2 Application to 32 Bits

With the mass of 16-bit legacy code out there, a lot of people are going to have to deal with the move to 32-bit operating systems very soon. Some of us in the OS/2 world have already undergone conversions, and some are still in the process. This article describes some of the pitfalls and practical issues involved in a mixed-mode and mixed-language project. Our project uses IBM's CSet/2++ product, so certain details and keywords might be different under other development environments.

OVERVIEW

Our company's product (a rather massive, sprawling, distributed Clinical Information System) is OS/2-based and currently implemented in 16-bit Modula-2. The company has just completed its first major pilot project toward moving the product to a fully 32-bit C++ platform. We had to deal with not only the issues of two memory models but also two different languages. We rebuilt our client user interface as a 32-bit C++ layer on top of a number of existing 16-bit Modula-2 DLLs.

Needless to say, much pain and effort was involved in understanding the issues and working around them. But before we examine those issues, we

need to take a short detour and discuss the fundamental concepts involved.

MANY MEMORY MODELS

Anyone who has done any software under DOS/Windows or 16-bit OS/2 knows what a nightmare memory models can be. A 16-bit application cannot (easily) create any single data structure larger than 64K in size. This might not sound so bad, but it is, because 64K is not really that large. To deal with larger data structures, the developer must do a lot of manual pointer magic. Some 16-bit compilers offer partial help via a HUGE model, but that help is extremely limited. So, many 16-bit applications tend to have a lot of seemingly arbitrary limits that are actually driven by the 64K barrier. OS/2 2.x, on the other hand, is a 32-bit environment; therefore, all addresses are just 32-bit offsets from address 0. Since OS/2 is a virtual memory system, each process sees its own virtual address space, which begins at 0 and goes up to 512MB.¹ The size of a data structure is limited only by the virtual memory available. Such a "flat" memory model is far simpler and more elegant than the alphabet soup of 16-bit memory models of DOS and Windows. Note that the flat memory model is effectively like the "tiny" 16-bit model (in



Dean Roddey

that everything is in one single "segment" and all segment registers can be set to the same value) except that its concept of tiny is orders of magnitude larger. So the flat model gets the speed benefits of a nonsegmented model but without the memory restrictions.

More importantly, this 32-bit flat model is the one that has been used all along by almost everyone except Intel, making such code portable to other processors.

The main concern in relation to the topic at hand is that pointers in each of these memory models are

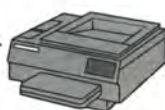
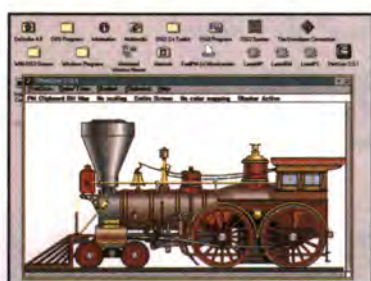
incompatible.² In addition, a 32-bit application can easily create data structures too large to be understood by the 16-bit world. A mixed-mode application must manually take on the responsibility of converting such pointers back and forth. Since the 16- and 32-bit compilers often cannot digest each other's prototypes, the developer must often provide external prototypes, which the compiler must take at face value. So you lose one of the greatest tools you have—the compiler's ability to flag incompatibilities. Our situation was further complicated since the 16-bit language was Modula-2. Compilers such as IBM's CSet/2++ offer mechanisms to convert pointers between 16- and 32-bit models, which we will discuss later.

ONE TOO MANY RUN TIMES

By far the worst problem to deal with is the issue of multiple run times. If you mix 16- and 32-bit code, even if they are both C, you must contend with the problem of two run-time library systems that both think they own the world. The problems we encountered here are too numerous to mention, but following are a few examples:

1. If your compiler supports multiple threads (as any respectable OS/2 compiler would), you must start threads via the language run time (instead of via the `DosCreateThread()` API provided by the system) because the run time must create per-thread data areas to store data such as error returns and the like. Also, the run time often will create an internal table of threads used to store thread status information. Now, what happens when your 32-bit threads (which were started via the 32-bit run time's `_beginthread()` function) call a 16-bit function? Who the heck knows? is the likely answer. Unless you have the 16 run-time source code, you can't know.

SINGLE KEY-STROKE SCREEN PRINTING



CLIPBOARD
VIEWER



SCREEN
SAVER

SCREEN CAPTURE
.WPG .MET .BMP .PCX .TIF .GIF .PCT



- Quality Color or Black & White copies of color Desktop Images.
- Copy any portion of any image on the Desktop.
- Screen Saver prevents monitor burn-in.
- Date & Time Display for Time-Stamping images.
- LAN installable. Complete support for LAN attached printers.
- Economical! 4 Utilities for 1 Price! Entity licensing available.
- The most stable and reliable product available.
- Includes both OS/2 2.1/2.0 (32-bit) and OS/2 1.3 (16-bit) versions.

THE LEADER: **PrntScrn**TM Screen Image Utility for OS/2



TM

MITNOR Software

Phone: (918) 357-1628

Fax: (918) 357-2869

SCREEN IMAGES IN OS/2 DEVELOPER ARTICLES ARE CAPTURED USING PRNTSCRN, COURTESY OF MITNOR SOFTWARE

Circle Reader Service Number 12

Actually we have the Modula-2 run-time source, and it's still hard to know because much of it is in uncommented assembly language. If you're lucky, the program falls over and you can explore the problem. If you're not, you cause some obscure error that is hours and megabytes away from the real problem.

2. If your language run times set up certain CPU and FPU registers and expect them to always be in a certain state upon entry to the program or to functions, how do you handle code that calls into and out of code with different expectations? Keep in mind that you can't always know when this happens if you have GUI code that spans the two worlds. Messages are kicked off constantly between windows that can cause transitions between the models. Catching every single transition and setting the flags would be totally impractical. A related issue is that the two run times might treat the FPU differently. Some might leave results on the FPU, while others might always expect the FPU to be empty. Some might even keep the FPU stack full all the time and use it like a set of registers. Incompatible use of the FPU is bound to cause major problems.

In our case, the Modula-2 run time's use of the FPU was of the later type. It filled the whole stack up (except one entry) and left it that way. It then manipulated the FPU registers as fixed-position registers for the most part. This was totally incompatible with the C mechanisms, so we reverted to the stack-based convention for our whole Modula-2 system, which prevented that type of FPU usage. It also prevented the Modula-2 system from generating more efficient, register-based calling sequences, however. And worse, it made the math libraries revert to an old DOS holdover method for

returning floating point values. Any procedure that returns a floating point value (in the register-based protocol) returned it on the FPU stack, which is a per-thread resource. In the stack-based system, a single, per-process variable is used to return it. Of course, this

is a potential disaster in a multi-threaded application since the return variable is not protected in any way.

3. What about floating point exceptions or exceptions in general? The 16-bit world did not have a really decent exception mecha-



MESATM 2

Spreadsheet Software for OS/2[®]



- Native OS/2 Spreadsheet
- Small • Fast • Powerful
- REXX scripting
- Objects to the core

SPREADSHEET TECHNOLOGY FOR A NEW FRONTIER.

**TO ORDER: CALL
1.800.315.MESA**

or contact your favorite
OS/2 reseller

\$199.
Suggested
Retail Price*



17 St. Mary's Court Boston, MA 02146
phone 1.617.734.6372
fax 1.617.734.1130

*Shipping & handling charges extra.

Circle Reader Service Number 13

nism. Exceptions worked on a per-process basis (and only on the primary thread), making multiple thread management of exception handlers a bit sticky. The 32-bit world fixes this problem and has a very elegant exception mechanism. But when, say, a math exception occurs, the run time that generated the exception expects to field the error and provide some corrective action. But if 32-bit code has installed exception handlers, it will also see 16-bit exceptions and has to decide whether it represents a serious error or something that is going to be fielded by the other run time and fixed.³

4. What if your 16-bit run time is a somewhat bad port of a DOS run time that makes all kinds of dangerous assumptions about its control of the environment? In our

case, the Modula-2 run time does things like storing values in magical places on the stack. Unfortunately, in a 32-bit world, the stack is managed in a totally different manner, and those magic places either do not exist at all or are in use already by the 32-bit world. During startup, our Modula-2 run time also uses a large chunk of the stack as a temporary work area; it stores the thread's ID there, whether the thread is in a math function that could cause an exception, and so on.

5. Then there is the problem of per-process initialization. The first example mentioned dealt with per-thread initialization, but both run times want to handle per-process startup. Unfortunately, it is highly unlikely that both of them can successfully do this without stepping

on each other's toes. In some cases, you might not even be able to make it happen because it is kicked off via some magical mechanism linked into the 16 .EXE that cannot be successfully linked into the 32 bit .EXE. So, `scanenv()`, for instance, would work in the 32-bit world but not the 16-bit world because its per-process init was never kicked off to set up this information. Most run times also depend upon the registers being in a predefined state (which is ensured by the operating system), but the first run time to init will destroy those initial contents. Having the source code, we have been able so far to selectively initialize the parts we need.

6. If, as in our case, you also are dealing with multiple languages, you have the extra problem of language constructs that are

Point and click your way to fast development of GUI client-server applications.

Do it with Guidelines, a second-generation object-oriented graphical workbench. Guidelines lets you create graphical client/server applications for OS/2 and Windows clients. A built-in prompter lets you simply point and click or drag and drop from the desktop using more than 30 presentation manager controls.

Guidelines utilizes JBA's high-level object-oriented language (JOT) to describe applications. After the application is designed, JOT automatically converts to C++ code, so you don't have to learn its complex syntax and rules. But if you already know C++ and want to use it directly, you can. The conversion takes place regardless of the platform intended.

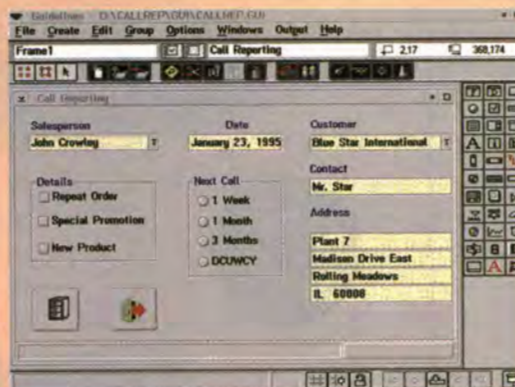
Guidelines saves time and simplifies the creation of graphical, platform-independent applications. Its scalable architecture lets you create anything from stand-

alone PC applications to complex enterprise-wide business solutions. Guidelines supports DB2/2, DB2/400, DB2/6000, DDCS, ODBC, Q+E Lib, and SQL.

Call for a free demo pack: 1-800-JBA-INTL.

In Canada: (905) 940-2442

JBA International
161 Gaither Drive, Ste. 200
Mt. Laurel, NJ 08054



© JBA International

JBA-111



Circle Reader Service Number 14

totally incompatible or available only on one side. For instance, Modula-2 sets have no equivalent in C/C++; Modula-2 commonly passes nonfundamental data by value, which C/C++ does not like to do; Modula-2 passes magic values that are not in the parameter list; enums are 1 byte on the Modula-2 side but are (usually) 4 bytes on the C/C++ side; and, of course, Modula-2 knows nothing of C++ objects.

One very problematic Modula-2 construct is the open parameter. This is a parameter that is passed by value but is of variable size. The compiler pushes a size parameter (that is not visible in the parameter list) that indicates the amount of data copied. C/C++ does not deal with the situation at all, so we had to provide alternate functions that

accept the parameters as VAR parameters and then call the original versions.

Other problems exist, but these are the major ones, and they should be more than enough to depress you. In many cases, we have been able to modify the Modula-2 run time to get around the worst problems. Of course, that means we have basically committed to using this version of the Modula-2 system forever (or until we get fully converted, whichever comes first). That's not as large a problem as it would seem, though, because our Modula-2 vendor is not exactly an OS/2 supporter. So it is unlikely that any significant improvements would become available even if we could easily accept them.

Without the source, we would have been totally out of luck, in

my opinion. But keep in mind that the run-time problem is not always going to be a problem. If the 16-bit code does not call any 16-bit run-time code (or a subset such as the intrinsic `memcpy/strcpy` stuff), none of these issues apply. But the amount of code out there that calls no run-time functions is small.

WHAT IS THIS DEBUGGING YOU SPEAK OF?

Written any massive software systems without a debugger lately? If you write a mixed-mode application, you probably will. Quite likely your 32-bit debugger will not have a clue how to trace into your 16-bit code. In our case, the 32-bit C/C++ debugger certainly does not know how to trace into 16 Modula-2 code. If you are developing a new 32-bit layer on top of



IF YOU KNOW OS/2®, YOU SHOULD KNOW INDELIBLE BLUE

Power Tools

chartParts	DAK25	\$209.00
Guidelines	JBA53	535.00
KASE:VIP	KAS44	495.00
C++/Views	LIA50	899.00
CA-Realizer	CAS20	89.00
VisPro C & C++	HCK56/55	175.00
SOMobjects Kit	96F8647	255.00
DCE Dev. Toolkit	96F8690	895.00
MKS Toolkit	MKS55	230.00
zApp for OS/2	IMK35	625.00
Object Factory	SAC40	229.00
Objectpm	RAL50/60	210.00
Q+E 5	QES50	344.00
AccessWPS	CRY30	42.00
IPF Editor	PCC84	135.00
Partition Magic		CALL!

Database

DB2/2 & DDCS/2		CALL!
OnCmd	ONL40	\$149.00
Watcom SQL	WAT88	295.00

VisualAge



by IBM

IBM's powerful new vision of programming! VisualAge is a client/server application development power tool that focuses on line-of-business applications. Components include visual programming tool, parts library, GUI support, client/server & communication support, enhanced DLL support, multimedia support and much, much more!

17H7495 Single User		
MSRP \$2,495		IB: \$1,550
17H7496 Team V2 OS/2		
MSRP 4,995		IB: \$3,100

Programmer's Editors

PREDITOR/2	COM78	\$149.00
RimStar Prog. Ed.	RIM20	239.00
SourceLink	SSI20	219.00
Qedit	SEM60	69.00
BOXER Text Ed.	BOX10	69.00
SlickEdit	MED10	149.00
KEDIT	MSG50	159.00

Programming Languages

IBM C Set++		CALL!
WatcomC/C++ ³²	WAT22	\$199.00
High C/C++	MET32	524.00
SmallTalk/V	DIG50	895.00
APL2 for OS/2	89G1556	125.00
FORTTRAN 77 ³²	WAT77	395.00
REXX, REXX & more REXX		
VisPro REXX Gold	HCK20	\$239.00
VX-REXX	WAT10	99.00
REXX SuperSet/2	GTU64	69.00
dbfREXX	DSF25	95.00
REXXLIB	QUR90	45.00

THE SINGLE SOURCE
FOR OS/2® SOLUTIONS

100'S OF OS/2 APPS AVAILABLE TODAY! CORPORATE DISCOUNTS AVAILABLE
CALL FOR OUR FULL COLOR CATALOG ORDERS: 1-800-776-8284



Circle Reader Service Number 16



legacy code, only a very small layer on the top will be directly debuggable. You have to do the rest via print statements and beeps. If you don't have perfect pitch, you will probably develop it quickly.

In our case, our new interface is replacing an existing interface that will be maintained for a while. This old, fully Modula-2 interface allows us to debug the 16-bit code within the 16-bit environment, but it does not help us understand why the same code does work correctly under the 32-bit interface. As time goes on, we must deal with the issue of maintaining the old interface purely for debugging, even though no one is paying us for it.

LETS DO THE THUNK

If you can get over all of the humps mentioned, how do you actually write a mixed-mode application?

You thunk. Thunking is what IBM calls it when 16-bit code calls 32-bit code, or vice versa. Thunking has to do two major jobs: get the calling sequence right and convert addresses to the appropriate format.

Both jobs are pretty easy if you use IBM's CSet/2++. Just by providing the correct prototype for an external 16-bit function, you tell the compiler how to do the correct calling convention and convert the addresses. Look at Figure 1 for an example of the way to provide the correct prototype.

The 16-bit function is declared as usual. If you used the `#define`d types in your 16-bit code, you don't have any worries about compatibility of your fundamental data types since, for instance, a `ULONG` is the same in both worlds. On the 32-bit side, an `extern` prototype is provided that tells

the 32-bit compiler about the 16-bit function's characteristics. The `extern` keyword says it is externally defined. The `"C"` attribute inhibits the name mangling that would normally happen under a C++ compiler. The `_Far16` keyword says the function is a 16-bit function in a separate segment (remembering that all 32-bit code is in one big segment.) The `_Cdecl` attribute forces a stack-based, C-type parameter pushing and stack cleaning scheme. It also forces the name to have a leading underscore since most 16-bit C compilers prepend an underscore to all function names. The first parameter (`ulCount`) is a fundamental type, so it needs no special attributes. However, it will be pushed on the stack as two 16 values as required by a 16-bit function. The second parameter is a string pointer, so it is given the `_seg16` keyword to say that it lies in a 16 segment. Note that the `_seg16` comes after the `*`, not before.

When the call to `ulFooCounter()` is compiled, the compiler will generate the needed code to push the parameters in the correct 16-bit way, push the return address as a segmented 16-bit address so the called 16-bit function's `ret` instruction will correctly pop the address off, convert the string pointer to the correct 16-bit format, and make an intrasegment call. So, there is not a terrible amount of work involved. However, there is no way for the compiler to tell you that your `extern` prototype is totally wrong, and it will happily call the external function incorrectly, causing major problems. Even if you get it right the first time, any future changes to the 16-bit code can go unnoticed, causing even stranger problems because you won't have any real idea where to start looking. You might conceivably come up with a conditional compilation scheme where the same header is included in both code bases with

```
// File C16Code.C
ULONG ulFooCounter(ULONG ulCount, CHAR* pszName)
{
    return .....;
}

-----

// File C32Code.C

// Create a prototype for the 16 bit
// function
extern "C" ULONG _Far16 _Cdecl ulFooCounter
(
    ULONG          ulCount
    , CHAR* _Seg16  szName
);

// Now call it
CHAR* pszName = "MyName";

main()
{
    printf("The count is %ld\n"
        , ulFooCounter(ulCount, pszName));
}
```

Figure 1. Calling a 16-bit C from 32-bit C.

different keywords being generated accordingly, but that would also have its associated maintenance nightmares given a large enough system.

What if one of the parameters is a structure that has many allocated substructures or maybe contains the headers to a couple of linked lists, which are lists of pointers to nodes? All of these addresses must be thunked for the 16-bit world to use them. In such cases, the 32-bit world often must effectively build a redundant copy of the structure, pass it to the 16 world, and then potentially copy all of the information back out into the original data structures. Needless to say, doing this for large data structures is both time- and memory-consuming. For a linked list, you could also conceivably build a separate node chain that pointed to the same node data to avoid duplicating the data itself, assuming that the data did not have any pointers that require conversion.

If your 16-bit code exports global data values, the same techniques are used to make it available to the 32-bit world. Of course, this assumes that the data is of a fundamental type and a type that is even understandable by the 32-bit code. As just discussed, if the global data value is a complex structure with a lot of suballocated pieces or a non-C data type, you either have to thunk each address each time you use it or go through the trouble of maintaining a separate copy and keeping them in sync. A better thing to do is generally just nip the whole globally visible data problem in the bud and not make it visible in the 32-bit world. If it must be accessed, provide access procedures to manipulate it and let the 32-bit world call them. If you must maintain global visibility, refer to Figure 2 for an example. I think you should also be careful about functions that do

not provide full prototypes of their parameters (that is, they have variable parameter lists, such as `printf`). These calls cannot provide explicit prototypes for their variable-length parameter lists, so you might have to explicitly cast pointer parameters to get the correct memory model.

LET'S DO THE MODULA-2 THUNK

In our case, we also have to deal with the fact that Modula-2 often passes things by value that C/C++ generally does not because Modula-2 (like plain C) does not have the concept of a constant parameter. So these values are passed by value to ensure the original data is not modified. Another problem involves open array parameters. The Modula-2 compiler copies a variable amount of data onto the stack and passes an array index limit to indicate the array's size. C cannot not handle that type of situation at all.

We dealt with this by providing shell procedures that accepted the same parameters (but by reference) and then called the original

procedures, returning its return value. This takes a lot of CPU and developer time to implement and muddies the architectural waters considerably, but it is the only reasonable way to deal with the problem. Luckily for us, our Modula-2 implementation is part of a multi-language development environment, so it provides pragmas for selectively controlling the calling convention of procedures. We used these pragmas to give all the shell procedures the 16-bit C calling convention.

A major pain comes from the need to define C versions of the structures on the Modula-2 side. You must have intimate knowledge of the data layout and packing system of both environments. For instance, enums can be either one or two bytes on the Modula-2 side, according to the pragma in use. Or, both environments allow packing to be controlled, and it is not always evident what packing scheme is in force. Or, how do you represent a Modula-2 set? You have to know how Modula-2 maps sets and creates a data type of the cor-

```
// File C16Code.C
ULONG  ulFooCount;
ULONG*  pulFooCountPointer = &ulFooCount;

-----
// File C32Code.C

// Create prototypes for the 16 bit data
extern _Far16 ULONG      ulFooCount;
extern _Far16 ULONG* _Seg16 pulFooCountPoInter;

// Now access the data
main()
{
    printf("The count is %ld, via pointer %ld\n"
           , ulFooCount
           , *pulFooCountPointer);
}
```

Figure 2. Accessing 16-bit data from 32-bit C.





rect size on the C side to cover the same area. Again, keep in mind that the compiler is utterly unable to tell you if these structures drift out of sync over time. You must do it manually and maintain it manually.

And then, to make matters worse, you have to define yet another version that is native 32-bit and use the first (translation) structure as a template to get the data out of the Modula-2 structure and into the native 32-bit structure. If you do not do this (that is, you allow the translation structure to be directly accessed from the 32-bit world), you are going to regret it later because you will have data dependencies all over the place. Also, you would lose much "typeness" in the process. For instance, since an enum is a byte (or two maybe) on the Modula-2 side, your translation structure must represent it as a byte (not as an enumerated type, which is usually four bytes on the 32-bit side.) And, just as a religious matter, you would want to hide the details of the old 16-bit world from the new 32-bit code so that it will not be affected when the underlying code is finally converted.

Another subtle gotcha is the global name space issue. Under Modula-2, all data and procedure names exist within the scope of a module, not in the global name space. This is similar to C++'s use of classes to divide the name space. When you are forced to use C calling convention on procedures in the Modula-2 world, you lose this name encapsulation. For linking purposes, the procedure name is then in the global name space and may suddenly conflict with other globally visible names. This is particularly likely because Modula-2 developers don't have the habit of prefixing names to make them more unique. They don't have to because the module name is an automatically applied prefix guaranteeing uniqueness.

SUMMARY

As you can see, doing a serious mixed-mode application is no trivial task. As previously mentioned, keep in mind that most of the really serious issues arise from the run-time problem. If you just want to call 16-bit TCP/IP libraries from 32-bit code, that should be very easy since such low-level code will most likely not have any significant language run-time calls. And, in many cases, the vendor will provide you with a header explicitly for a mixed-mode programming, which has all of the correct prototypes. But, if you are porting a large project of custom code, it goes without saying that it will likely make heavy use of 16-bit run-time code for some period of time. The seriousness of the problems is really related to how dissimilar the two environments are, with different languages being kind of at the top of the scale.

Sure, a person has to do what a person has to do, but don't undertake it lightly. Do some experimentation and think about investing the time in developing tools to au-

tomate some of the grunt work involved. If your system has a clean separation across some sort of interprocess communication, you can port one side at a time without having any mixed code in the same process. If you do go with the mixed mode system, you might have to dedicate a developer or two just to manage and document the intraenvironment wasteland to keep it from becoming a massive disaster. In my opinion, you might actually spend less and disappoint fewer customers in the end by dedicating a few developers to a parallel development project that starts from scratch; but that view might be a little long for most companies who often must justify the port via some immediate results.

Dean Roddey is a senior engineer at Quantitative Medicine, Annapolis Md., A Marquette Company. Dean loves computers, a lot of coffee, and minimal physical activity, so enjoy his articles while you can. He can be reached on CompuServe at 72170,1614 (72170.1614@compuserve.com via Internet) or in the CompuServe OS/2 Developer forum.

ENDNOTES

1. The theoretical limit is 4GB, of course, because of the 32-bit addressing. But OS/2 was purposefully limited to 512MB to greatly increase the ability to run 16-bit applications. Some hardcore folks deride this limitation, but it is not a problem for most of us.
2. Their contents are not only incompatible but also have different alignment requirements for the data they point to. To get around this under CSet/2++, use the /Gt+ compiler option to generate tiled memory. Basically, it ensures that any 32-bit memory object is allocated in a way that ensures its address can be converted correctly between the two environments.
3. As far as I can tell, the 16-bit caused exception travels up the handler chain like any other 32-bit exception and is seen by any 32 bit handlers installed on the call chain to the 16 bit call. If the exception makes it to the top level handler provided by the system, the exception is translated to a 16-bit signal type exception, if applicable, and dispatched to the 16 handlers, if any.

You Discovered the 32-Bit Power of OS/2.

FOR CORPORATE
DEVELOPERS BUILDING
SOLUTIONS ON
OS/2!

Now Master It at OS/2 DEVELOPMENT '95.



Introducing the conference for advanced software development.



he publishers of OS/2
Developer magazine have
assembled industry leaders to host the first
independent Pan-European conference on
OS/2. You'll sharpen your OS/2 skills
and learn from the experts about the latest
techniques and technologies to increase
your effectiveness on the job.

What's included:

- Excellent educational seminars
- Keynote speakers
- Product Roundtables
- And special events!



You'll Learn More About

- SOM programming
- Workplace shell programming
- User interface development
- OpenDOC
- Object-oriented programming
- REXX programming
- SmallTalk, and more!

Produced by

**OS/2
Developer**
magazine

May 8-10, 1995
Krasnapolsky Hotel
Amsterdam

To find out more
about this exciting
OS/2 event, fax,
phone, write, or
contact us by
electronic mail
now!

Telephone:
32-2-538.60.40
From the US:
011-32-2-538.60.40

CompuServe:
76307,1054

Internet:
pwesterman@mfi.com or—
[http://www.mfi.com/os2dev/
amsterdam.html](http://www.mfi.com/os2dev/amsterdam.html)

Mail:
OS/2 Development '95
Chaussée de Charleroi 123a
Bte 5
B-1060 Brussels, Belgium

Miller Freeman, Inc.
A MEMBER OF THE GRIFFIN PUBLISHING GROUP

OS/2
DEVELOPMENT '95

**SEND BACK THIS
COUPON TO
FAX 32-2-537.56.26
FROM THE US:
011-32-2-537.56.26**

Name

Title Company

Mailing Address:

Telephone

Fax Email



Targeting both OS/2 and Windows is tough work, but their similar interfaces greatly ease the development burden. **By BERNIE THOMPSON**

Warp and Windows: How Similar, How Different

OS/2 and Windows are sibling systems. They share the similarities of family, yet there are many frustrating differences. Most developers have felt they couldn't overcome these differences and produce an OS/2 product. But the dynamics of the software industry are changing. Microsoft has developed control over critical software standards. It has leveraged these advantages to dominate a large part of the mind share and sales of the software industry.

Microsoft has chosen not to compete in the OS/2 market they originally helped create, however. This makes OS/2 Warp an interesting opportunity for all other software developers.

In the years since IBM took over development, OS/2 has made great technical advances. The Workplace Shell is unrivaled as a dynamic, functional, elegant user interface. OS/2's symmetric multiprocessing abilities have no counterpart in Windows 3.x or Windows 95. And OS/2's other innovative features and position as an alternative to Windows have produced a steadily growing and committed user base.

Developers can harness this solid OS/2 market by producing native applications offering advantages over their Windows counterparts. Because the OS/2 programming interface is of the

same family as the Windows interface, porting doesn't need to mean rewriting.

It is important to understand the differences between Windows and OS/2 so you can write the core of your application with both systems in mind and then add OS/2-exploiting features onto this base. This is certainly a demanding task, but the rewards will justify the effort when your product stands out in the OS/2 market.

PROGRAMMING FOR PORTABILITY

The best way to produce a cross-platform application from scratch is to build portability right in. That's possible by using object libraries such as Taligent's CommonPoint framework, Borland's Object Windows Library, Zinc's C++ framework, free systems like YACL, and many others. As the diversity of hardware and software systems increases, use of these libraries becomes the obvious choice for new application development.

Unfortunately, not every developer has the luxury of writing applications from scratch. Many developers need to take an existing application and evolve it into something more portable. And even for those who use frameworks, it is sometimes necessary to go straight to the native API for reasons of perfor-



mance or functionality. So for all developers, it is important to understand the nuances of the underlying operating systems as much as possible.

KINDRED APIS

In the late 1980s, Microsoft envisioned Windows as a stepping stone to OS/2. Design decisions made in both OS/2 and Windows still reflect that strategy. This makes OS/2 an inviting choice for those looking to port their Windows programming skills and applications.

At an abstract level, the two systems are almost identical. The primary system-wide object is the window. A window sends and receives messages to do its work. The system provides some standard window types for all applications to use (menus, scroll bars, and so on). These windows can be subclassed to extend or modify their functionality.

But below the abstract level, the systems have many differences. The following sections take a closer look at a sampling of the problems and bright spots. The details have gotten somewhat more complex with Microsoft's new Win32

APIs. The following information is a mix of Win16 and Win32. The bright side is that Win32 has actually brought Windows closer to OS/2 in many significant ways.

SOME DETAILS

Windowing. In both systems, a window is created by registering a window class and creating an instance of that class. Messages are then received and processed by a window procedure.

When registering the window class, do not rely on class words. OS/2 doesn't have them. Also, Windows allows assignment of default icon, cursor, background brush, and menu resources when registering the window class. OS/2 deals with these resources on a per-window basis as each window is created. For this reason, it may be convenient to bundle window registration and creation into one procedure and produce one version for each platform.

OS/2 programs generally use `WinCreateStdWindow()` to create the main window. This is equivalent to `CreateWindow()` with `WS_POPUP` style in

<i>OS/2</i>	<i>Windows</i>	<i>Comment</i>
<code>WinInitialize</code>		Automatic in Windows
<code>WinCreateMsgQueue</code>		Automatic in Windows
<code>WinRegisterClass</code>	<code>RegisterClass</code>	
<code>WinCreateStdWindow</code>	<code>CreateWindow</code>	
<code>WinShowWindow</code>	<code>ShowWindow</code>	
	<code>UpdateWindow</code>	Automatic in OS/2 after shown
<code>WinGetMsg</code>	<code>GetMessage</code>	
	<code>TranslateMessage</code>	Automatic in OS/2
<code>WinDispatchMsg</code>	<code>DispatchMessage</code>	
<code>WinDestroyWindow</code>		Automatic in Windows
<code>WinDestroyMsgQueue</code>		Automatic in Windows
<code>WinTerminate</code>		Automatic in Windows

Table 1. Basic flow of application creation/destruction.

Windows. Windows `WS_CHILD` style is equivalent to calling `WinCreateWindow()` in OS/2. Table 1 illustrates the flow of application creation and destruction in the two systems.

On both platforms, a main window has two basic parts: the frame (called the nonclient area in Windows) and the client. The client area is the main area of the window where the application does most of its work. The frame surrounds the client window and provides a set of standard functionality: the system menu, a title bar, a minimize and maximize button, a sizing border, and more.

In Windows, the client and frame are bundled together and treated as one window. In OS/2, they are separate. The frame is a standard control, just like any other.

How can Windows bundle these two windows together? It sends both sets of messages to the one window. Messages intended for the frame will have the prefix `WM_NC` (NC stands for nonclient). Messages for the client will have the normal window prefix of `WM_`. In OS/2, there are no special prefixed messages; they just go to the proper window. In effect, every Windows window is made to subclass its frame, with the messages going to a single window procedure. In OS/2, the program must explicitly subclass the frame if it

wants to change the handling of frame messages.

Consider putting all `WM_NC` handling into a single function. When in Windows, call this function from the main window procedure. In OS/2, redefine `WM_NC` messages to their `WM_` counterparts, and use this function to subclass the frame.

MESSAGES.

Windows and OS/2 messages consist of a window handle, a message ID, and two effectively untyped parameters. The real data that comes with the messages is encoded in the fields of these two parameters.

In Figure 1, you can see how Win32 messaging has moved closer to OS/2 in terms of parameter size. Although the sizes are identical, Win32 does encode this data differently than OS/2. In fact, Win16 sometimes encodes it differently than Win32. This causes a large portability problem.

Using macros to extract the data for you is a good solution. Windows programmers already have a version of these facilities in the form of the `windowsx.h` header in the Windows toolkit. This header, intended in part to help portability between Win16 and Win32, provides facilities for unpacking all data from a message and calling a function to handle it. Windows

calls these macros "message crackers." By producing their own application macros for Windows and OS/2, developers can provide a foundation for portable message handling.

Since most interface work is done through message passing, this will probably be the most difficult area of a porting effort. Just counting system messages (not controls), Windows and OS/2 have roughly 150 different messages each. The parameters of each message have to be analyzed as if it were a normal API. Then the application programmer must find a way to work within the subset of function provided by both systems. Fortunately, this subset is large and functionally rich.

GRAPHICS

Possibly the most famous difference between Windows and OS/2 is the coordinate system. OS/2 operates in Cartesian Quadrant I with (0,0) in the lower left-hand corner. Windows operates in Quadrant IV with (0,0) in the upper left-hand corner. Figure 2 illustrates this difference. Programs should use macros and functions to hide coordinate differences wherever possible.

Windows uses drawing tools to represent drawing attributes. Brushes determine a fill pattern. Pens determine a line pattern. OS/2 can use area and line bundles to provide these functions.

OS/2 uses Adobe Type Manager Type I outline fonts. Windows uses TrueType outline fonts. The standard font types will vary from system to system. Font selections must be converted to the nearest available type.

Advanced graphics facilities, like Bezier splines and transforms, have long been supported by OS/2. Some Windows platforms are now adopting these features. Both Windows NT 3.5 and

Win16	Win32	OS/2
HWND	HWND	HWND
MESSAGE	MESSAGE	MESSAGE
WPARAM	WPARAM	MPARAM
LPARAM	LPARAM	MPARAM

Figure 1. Message layout.

Windows 95 support Bezier splines, and Windows NT 3.5 supports transforms. Portable applications can now take advantage of these powerful OS/2 features.

Metafiles are supported by both systems. Metafiles are effectively playlists of graphics drawing functions. This means that unless the drawing APIs supported by two systems are identical, the metafiles will not be compatible. Since OS/2 and Windows have different graphics APIs, metafiles for one system cannot be played on the other. Even metafiles from Windows NT can't always be played on Windows 95. This precludes using metafiles as a cross-platform data format.

An interesting new twist in graphics is the OpenGL 3-D library. This library is now available for OS/2 through IBM's Developer Connection and for Windows NT in version 3.5. This presents an exciting new opportunity for developers. If OpenGL is suitable for the application, it can be used instead of the native graphics APIs. This can help solve the graphics porting dilemma.

MULTIPLE DOCUMENT INTERFACE

During the development of Presentation Manager (PM), IBM considered implementing multiple document interface (MDI). Research showed, however, that MDI caused significant confusion among novice users. MDI went on to become a standard Windows paradigm, while OS/2 resisted the feature. Recently, Microsoft has realized MDI is a usability problem, and it has decided to avoid using it in its own Windows 95 interface. Applications should use a document-centric paradigm instead of MDI.

CONTROLS

Controls are a large issue in themselves. For the core controls (but-

ton, listbox, and so on) the best approach is to use the functionality provided by OS/2 and then map it to Windows. This is because Windows provides a superset of OS/2's functions for these controls.

The newer controls are a more difficult issue. OS/2 2.0 introduced many new, exciting controls that provided a basis for the functionality of the Workplace Shell. Microsoft has adopted many of the ideas in these controls for Windows 95. Developers should carefully pick functions that are common between the two platforms or avoid these new controls. Table 2 shows how Windows 3.1 and Windows 95 controls map to OS/2.

COMMON DIALOG BOXES

Both systems provide standard dialogs for selecting files and fonts. Other dialogs are not available on both systems. Applications must provide their own dialogs to cover for missing common dialogs.

MEMORY MANAGEMENT

Memory management differences have receded between OS/2 and

Windows with Win32. Win32 and OS/2 both have 32-bit designs that closely mirror the functionality provided by CPUs like the Intel 386. Only a few Win32 features need to be avoided. Don't request specific virtual address ranges, as Win32 allows. And don't query or modify the page permissions of other processes. Both of these features are incompatible with OS/2's memory scheme.

For most cases, simple `MyAlloc` and `MyFree` functions, which wrap Win16, Win32, and OS/2 memory management, are acceptable ways to cover differences.

The best portability solution is to stick with the C and C++ mem-

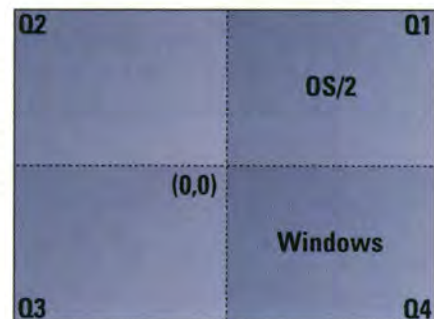


Figure 2. Graphics coordinates.

Windows	OS/2
Button	Button
Combo Box	Combo Box
List Box	List Box
Menu	Menu
Scroll Bar	Scroll Bar
Static	Static
Edit	Entry/MLE
Column Header (Win 95)	Container
List View (Win 95)	Container
Progress Indicator (Win 95)	Slider
Property Sheet (Win 95)	Notebook
Rich Text (Win 95)	None
Status Bar (Win 95)	None
Tool Bar (Win 95)	None
Tree View (Win 95)	Container
Spin Box (Win 95)	Spin Button
None	Frame
None	Value Set

Table 2. Common control mappings.

ory management functions whenever possible. These will guarantee compatibility with current and future operating systems.

THREADING AND SEMAPHORES

OS/2 users have rejected many applications that do not take advantage of native OS/2 functionality. In particular, multithreading is a native OS/2 feature that produces more responsive applications. This has been a great dilemma for developers because applications must be designed with multithreading in mind to make proper use of it. Since Windows 3.1 had no multithreading, portable applications could not be designed to use it.

This is another area where Win32 has brought Windows closer to OS/2 and opened the

door for more portable applications. Win32 supports threading and semaphores in a manner similar to OS/2. Now developers can create multithreaded applications and still port them between Windows and OS/2.

ATOMS

Atoms are nearly identical in the two systems, except for one important detail: in OS/2, atoms are case-sensitive. In Windows, they are case-insensitive, which impacts all system functions that use atoms in the underlying implementation. Most important of these is the class name specified in (Win)RegisterClass(). Fortunately, it is easy for the application to work around this difference by keeping case consistent within the application code.

RESOURCES

Name all resources with an integer ID. OS/2 doesn't support naming resources with strings. This rule can be followed by always using the MAKEINTRESOURCE macro provided in Windows. Produce an OS/2 version of the macro, then use this macro when identifying all resources. The resources themselves must be converted between the two platforms. The SMART porting tool, available on IBM's Developer Connection, can automate much of this task.

OPENDOC AND OLE

Technologies for producing software "parts" propose to revolutionize the whole software industry. That vision has yet to be achieved by any one technology. OLE reached the software indus-



OS/2™ is a great operating system - except when it comes to delivering a consistent desktop time after time. It's so easy to drag and drop system files and utilities into the shredder. Anyone can do it. Now that's an administrative nightmare.

Unless you have the **Desktop Observatory™** from Pinnacle Technology. The Desktop Observatory silently takes control of your user desktops.

Drag and Stop.

Lock & Load...Over a LAN!

Drag the OS/2 System folder into the shredder? No problem. The Desktop Observatory can dynamically rebuild their desktop from a small configuration file you control, and everything is restored. Store the desktop configurations on your LAN, and you can instantly update hundreds of desktops from a central location.

Security & Auditing

But there is even more to the Desktop Observatory. Apply restriction settings to objects so that they cannot be moved, copied, deleted, renamed, or even seen. Password protect folders and programs, and restrict access to files, folders and directories.

You can even start REXX or C utilities when folders or programs are opened or closed - great for backups, statistics collection, audit control, and copying files to or from a server.

Now you've found the perfect solution for OS/2 security, management and control - the **Desktop Observatory for OS/2.**

1-800-525-1650



Desktop Observatory is a trademark of Pinnacle Technology. OS/2 is a trademark of IBM Corporation. ©1994 Pinnacle Technology • P.O. Box 128 • Kirklin, IN 46050 • 317.279.5157

Circle Reader Service Number 15

try first. It is central to Microsoft's operating-system strategy. OpenDoc has come second, but with technology that most observers consider superior. Even more important, OpenDoc is more open. It is managed by a nonprofit organization funded by a large set of companies.

For developers taking a long-term view, OpenDoc is clearly the better choice. For those concentrating on the short term, neither technology provides a solution for all the software platforms available today.

STRUCTURING FOR SUCCESS

These many differences and issues may seem daunting. But with a properly structured application, the workload can be greatly diminished. The number one porting

rule for any program is to isolate platform-dependent code. In today's GUI-centric world, that means isolating the interface from the real work an application does.

A classic example could be a ray tracing program. One way to design such a program would be to mix editing, tracing, and image display together in the window procedure of a program—but this would make porting a nightmare.

The better approach is to have an editor program that passes its output to a tracing program (which has no interface code), which then passes its output to a display program. Now the core functionality of the application (ray tracing) has the potential to compile without any porting at all. By properly hiding what's going on behind the scenes, the program

can act exactly as if it were written as one single piece of code.

WHERE TO START

Obviously, the information in this article only scratches the surface of the Windows/OS/2 porting story. For large, complex applications, porting can be very involved.

Two good sources for further information are Charles Petzold's Windows and OS/2 programming books, *Programming Windows 3.1*, 3rd ed. (Microsoft Press, 1992) and *OS/2 Presentation Manager Programming* (Ziff-Davis Press, 1994). Many developers first learned to program Windows with Petzold's book. In 1994, Petzold released a new version of his OS/2 PM programming guide. Many chapters in these books parallel each other. Both books have a large



**TechBridge Builder™ — the
first framework-oriented
visual programming tool
for OS/2 — opens the door
to the next generation
of software development.**

TechBridge Builder's secret weapon is its framework of 700+ carefully honed object building blocks. It's like having a PM assistant, an object technology guru, an SQL expert, and a CUA advisor right in the box. The framework takes care of the tedious technology details while you concentrate on meeting user needs.

Demo disk available. Call 1 (800) 463-8998

TechBridge Technology Corp.

5001 Yonge St., Suite 1301, North York, Ontario, Canada M2N 6P6. Phone: (416) 222-8998. Fax: (416) 222-0168.

TechBridge, TechBridge Technology, and TechBridge Builder are registered trademarks of TechBridge Technology Corp. All other product names are trademarks of their respective companies.



number of sample programs, and many of these samples have both OS/2 and Windows versions. Developers working in both environments would do well to use the books as resources to compare the two platforms.

Many issues in porting between Windows and OS/2 are unique, but not the general strategies. Developers have long since developed strategies for porting between Windows and Macintosh, even though these systems are not as similar as OS/2 and Windows. Steve Petrucci's book *Cross-Platform Power Tools* (Random House, 1993) explores approaches for porting between Macintosh and Windows. The ideas he presents can provide an excellent basis for a Windows and OS/2 porting effort.

For those writing applications from scratch, don't fail to investigate cross-platform libraries. Wade Guthrie's *Portable GUI Development Kits FAQ* (comp.answers, comp.windows.misc.) is one of the best sources of comparative reviews and information on these libraries.

Anyone porting a Win16 or Win32 application to OS/2 should get the SMART porting tool mentioned earlier. This tool analyzes and partially migrates source code and is a great source of information in any case.

PLANNING ON CHANGE

OS/2 and Windows have a long history of symbiosis and rivalry. Before Windows took the market share lead, the industry believed OS/2 owned the future. Now the industry believes Windows owns the future. But every industry observer should know it won't stay that way forever. Developers should invest in the future of their code by investing in portability. As

the market makes its inevitable twists and turns, companies with portable applications can be the first to react. Portability is a powerful advantage and a smart strategy.

Bernie Thompson is an associate programmer with the IBM OS/2 Presentation Manager development team in Boca Raton, Fla. He has a B.S. in Computer Science from Penn State. Bernie can be reached at bthompson@vnet.ibm.com.

REFERENCES

Call 1 (800) 288-5545 for more information about Taligent CommonPoint.

Call 1 (800) 336-6464 for more information about Borland's OWL.

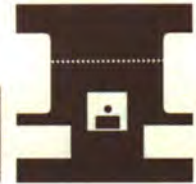
Call 1 (800) 638-8665 for more information about the Zinc framework

YACL—Yet Another Class Library, developed by M.A. Sridhar at the University of South Carolina. Available for Windows, Unix, and OS/2. <ftp://ftp.cs.scarcolina.edu/pub/sridhar> for source code and more information. Look for the book *Developing Portable C++ Applications using YACL* from Addison-Wesley in 1995.

Adrian King, *Inside Windows 95*, Microsoft Press, 1994, p. 190.

Call 1 (800) 6DEVCON to order IBM's Developer Connection.

Windows, Windows 95, Windows NT, and OLE are trademarks of Microsoft Corp. OS/2 Warp, Taligent, and Presentation Manager are trademarks of IBM Corp. Zinc is a trademark of Zinc Software Inc. OWL is a trademark of Borland Inc. OpenDoc is a trademark of the Component Integration Laboratories.



This article discusses the reasons to port to OS/2 and gives an overview of the various methods—past and present. By DERREL BLAIN

Porting Applications to OS/2: A Historical Review of OS/2

In the business of application development, changes in underlying technology happen rapidly and are apparently unending. Computer platforms, machinery, operating systems, storage subsystems, and coding tools constantly change. In the best of all computing worlds, applications could be recoded from scratch every time to take advantage of this new platform. Developers would have a lot of time to meditate on every single machine instruction. However, economic reality demands that we get things out the door as fast as possible. This urgency of the market creates a need for a porting strategy other than a careful recoding, one that involves using as much of the old application code as possible.

PORTING FOR CUSTOMERS

In practice, porting affects two groups: customers and developers. The desires of the two are different, and both must be accounted for in porting an application. Customers that have experience using your application, especially those with large installed bases they must train for and maintain, expect the application on the new platform to perform much like the previous version. They do not want to relearn how to do something they already know. They certainly

do not want hiccups and bugs. Something about the new computer hardware, the new operating system, or the new network has caused them to make a change. They want to see value in that change. That sense of value includes your application. If your application does not perform as well as expected and the customers believe in the choices they made for new underlying technology, they will find fault with your application. They will blame you.

PORTING FOR DEVELOPERS

Application developers, on the other hand, are generally driven by the urgency of the market rather than technology for its own sake. They would be happiest if they could have their applications magically work on the new technology without any changes to their code. This, of course, does not happen often.

In some ways, the ultimate in portability was dear old C code. At the application level, when practically everybody used some form of text screen interaction, an application could be maintained on several platforms simply by using a different compiler. The compiler was the porting tool. Reuse of the code object was practically 100%! One of my first jobs in the computer field involved assisting with maintaining C code libraries

for text windows. This code worked in both UNIX and DOS without much trouble. Even further back, programs I wrote for my own use on a CP/M card and crammed in my old Apple machine had no trouble porting to my shiny new 8086, despite the fact that the underlying processor and operating system were completely different.

So for the user, porting means some identity of use. For the developer it means identity of code—the closer to identical, the better. Given our practical definition of porting, we must ask ourselves, Is it even reasonable to think about porting applications between Windows and OS/2? Certainly! Before I can make a case for that answer, however, I have to lay some other groundwork. Other questions come to

mind that must be answered, such as: Why can't we simply recompile our application code and have it run on either Windows or OS/2? Why are things different now? What are the mechanisms involved that either help or hinder our effort?

The first answer is that the platforms of today are more complicated. Yes, systems are more complicated, but this is not where the difficulty lies. Because if the complications of the software, the hundreds of details that need to be worked out in a large application, have already been completed for the original application, then they have been completed for the ported application. Again, the user expects the same functionality, and the developer expects to reuse the code. So where does the difference enter? Whence the complication?

The difficulty is caused by changes in the system services.

The uniformity of the run-time libraries and system services like disk and text I/O was one of the things that made C wildly popular. Besides being useful shorthand for writing an algorithm, the system libraries largely agreed and worked in a uniform manner across many different platforms. Similar services existed throughout that technology space.

Today, despite the hoopla, the underlying structures on which applications are based have not changed much—even in this brave new world of graphical user interfaces. Programmers still work with more or less linear algorithms with some loops and some branching. Generally we use some form of structured programming with a main program relying on any number of functions or subroutines. For data, we basically still use things like tree structures, arrays, and lists. Memory, whether protected or unprotected, is still a linear space that is filled from the top and the bottom toward the middle.

WHY WINDOWS OR OS/2, ANYWAY?

Just for fun, consider this assertion: Microsoft's Windows and IBM's OS/2 are nothing more than big, fat third-party coding libraries. This does not mean they are not good. They accomplish many tasks that would take much longer and require more staff if a development team was going to code from scratch. However, the use, the task, and the underlying functionality your application brings to the platform is still your construction, your algorithm. What you gain as a developer from Windows and OS/2 is simply access to a bunch of library code, generally dealing with display and user interaction, which is beyond your immediate ability to create and debug.

Take control of your OS/2 scheduling needs with Advanced Task Scheduler for OS/2

September			1	2	3
4	5	6	7	8	9
11	12	13	14	15	16
18	19	20	21	22	23
25	26	27	28	29	30

ATS

Version 3.0



Are you looking for a way to manage your production job streams? Do you have programs that need to run after the completion of one or more other programs? Do you want to run programs on a regular basis? Do you want to process files that have just been received from another system or modified locally? Do you need to take action if a file is missing? Do you need to schedule around holidays and periods of heavy CPU load?

With ATS you can run your programs when YOU want them to run with out you having to be there.

Upgrades Available
from Version 2 and
other Task Schedulers
Call or E-Mail for Details

Here are some of the features of ATS for OS/2:

- * Manage Production Job Streams
- * Programs can be scheduled to run anytime.
- * Programs can be dependent upon Files, Other Programs, Holiday Schedules, and External Signals
- * Integrated Operators Console
- * Logging - To File and Console

New Features:

- * Job Queues - for managing your resources while managing your tasks
- * Periodic Scheduling
- * COMPLETE API and Command Line Interface

MHR

Software and Consulting
2227 U.S. Highway #1 * Suite 146
North Brunswick, NJ 08902

Voice: (908) 821-0359 * Fax: (908) 821-0350 * CompuServe 70312,627

Only
\$349

Circle Reader Service Number 18

You use the API entries of these GUI operating systems in the same manner as you use other coding libraries to further your own application development. Tasks that are uniquely yours are coded from scratch. Code somebody else has already written and debugged that gets something useful done for you, you buy and include in your own application. This is exactly how third-party coding libraries are used.

The architectures of Windows and OS/2 are similar to each other. With more time and space, I could make a convincing case that they are functionally identical. They both rely on event-driven messaging architectures. They are both windowing systems with parents and children arranged in a hierarchy. Their printing and graphic subsystems are extremely similar. Even the pieces of their applications, resources, dialogs, icons, and help systems are practically identical in use and similar in form. There are many reasons for this, some having to do with how logical it is to perform a certain task in a certain way. But the chief reason they are so similar is that Windows and OS/2 have the same mom and dad, IBM and Microsoft. Windows and OS/2 are parallel systems that grew out of the same fundamental architecture that was shared between the design teams of both companies. It is this similarity and the fact that developers on both platforms still code in some variant of C that makes porting between the two platforms possible

WHY PORT TO OS/2?

Before we take up the issue of what strategy to use for porting, I want to say a brief word about the reasons to port. An application should be ported when a company believes it can reach a profitable customer base. This is the only reason. No other issue is important. All this

endless discussion about the relative merits of OS/2 over Windows, or vice versa, is simply not useful for the application developer. Although it might be fun, and it is certainly good for spurring both companies on to making even better operating systems, your only concern as a smart businessperson should be the economic viability of your particular application on that platform. Three years and several million copies later certainly shows OS/2 to be viable.

The similarities between Windows and OS/2 can easily be seen in the history of the three tools that have been used to port from Windows to OS/2. These are the Microsoft Windows Libraries for OS/2 (WLO), Micrografx's Mirrors, and the Source Migration Analysis Reporting Toolset (SMART) from One Up Corp.

SMART is the most recent tool. Mirrors became available with OS/2 2.0 and was available as a porting tool until just recently. The Microsoft WLO library was the earliest, first available in the OS/2 1.21 time frame.

MICROSOFT WINDOWS LIBRARIES FOR OS/2

Believe it or not, Microsoft itself once had a strategy for helping developers reach OS/2, but WLO never saw wide distribution. In fact, the manual I have lists itself as version 0.9. WLO was essentially a library against which the Windows application was linked. The application to be ported continued to function as a Windows application. The similarities between this manner of operation and the Mirrors conversion tool are obvious; however, WLO tilted

OS/2
WINDOWS
WIN 95
WINDOWS NT



THE INTELLIGENT PROGRAMMER'S EDITOR



SOFTWARE
Development
PRODUCTIVITY
AWARD
1993



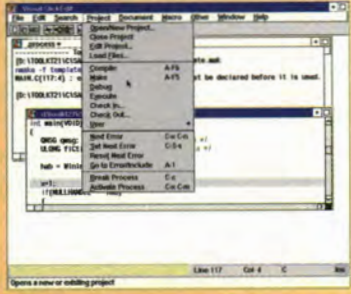
WRITERS
ASSURANCE

Visual SlickEdit™ is the high powered programmer's editor with the speed, ease, and features you need to ignite your productivity. Learn in no time with our comprehensive CUA, Brief, Emacs, and VI emulations. Take off with our extensive language support, project management, and NEW SmartPaste™.

Powerful features:

- Incremental search/search and replace
- Completely configurable
- Spell check comments and strings
- Typeless, object-oriented, C-style macro language
- Built-in dialog editor
- Clipboard Inheritance™ (patent pending)

Visual SlickEdit supports C/C++, Pascal, Fortran, Basic, dBASE, Modula-2, Assembly, COBOL, and Ada. The Windows NT version is available for Intel, Mips, and Alpha AXP machines.



Cut your work in half using:

- Procedure tagging
- Syntax color-coding, expansion, and indenting
- SmartPaste reindents pasted source code according to nesting level
- Compiler error processing

To Order, Call 800-934-EDIT also (919) 831-0600 or FAX (919) 831-0101

Visual SlickEdit, SmartPaste, and Clipboard Inheritance are trademarks of MicroEdge. Windows NT, Mips, and Alpha are trademarks of their respective manufacturers.

MicroEdge, Inc. PO Box 18038, Raleigh, NC 27619-8038

Circle Reader Service Number 19



more toward dynamic conversion than Mirrors did. Rather than converting those things that needed to be converted up front, WLO itself would dynamically convert things as the application was running. At that time, OS/2 still directly supported loading Windows-executable files. Windows and OS/2 also shared the same executable file format. The loaders on the two systems manipulated the application resources in a similar manner. WLO directly loaded and displayed Windows resources, so no changes were required.

With WLO, Microsoft converted the Windows Help Manager, WINHELP.EXE, so it would run directly in the OS/2 environment. This meant that help files that belonged with the ported application could be used directly. This strategy would not work as well later when the appearance of the two platforms diverged.

We also see in this initial effort the first mention of issues that still concern developers who port applications. These include warnings about changes in message ordering, reliance on low-level system calls, and changes in the underlying file systems, for example. However, each of these things, while bothersome, are not overwhelming technical issues to surmount. Low-level calls should be replaced with higher-level API entries. Programs can easily be re-coded to accommodate varying name lengths. The message re-ordering, while more troubling, can be accommodated by changing the response of the application to certain messages.

MIRRORS

Mirrors was released with OS/2 2.0. Mirrors was always intended as a short-term marketing strategy for getting developers to put their applications on OS/2 2.0 right away. Mirrors shared much of the

same strategy as WLO. By then, OS/2 had made significant changes, and Mirrors as a porting tool reflected this divergence. Mirrors, as is often the case with a piece of software, took on a life of its own and continued far past its expected term.

Although OS/2 2.0 had made many significant advances, porting from Windows in a fairly automatic way was still possible. Mirrors consisted of a DLL that the ported application was linked against, just as WLO was. Where WLO remapped and forwarded the Windows calls made by the application to OS/2, Mirrors satisfied many of the calls internally. Some were forwarded, but many others never left the Mirrors DLL. So although it was called an emulation layer, Mirrors did not in fact function strictly as an emulation layer. For example, applications would often select a brush many times before it was actually used. Mirrors would not realize that brush in the OS/2 environment until immediately before its use. Structures, lists, instance information, and much more were maintained internally to the Mirrors DLL. Essentially, Mirrors became the Windows system libraries for the ported application.

Rather than incur the overhead of dynamic conversion for resources, help, icons, and so forth, command line utilities were developed to convert these things up front. Mirrors had its own resource compiler. The help files for the ported application were also converted by a command line utility ahead of time. This meant that when the resources were bound to the application's .EXE (now an OS/2.EXE), they did not need to be converted or mapped at run time, which made the whole application actually much closer to an OS/2 application.

If something as large as a help

system, with all its fonts and embedded bitmaps and links to an application, can be converted with a simple command line utility, which MIRHLP is, then the real systemic differences between these two operating systems is not large. This point is important because it demonstrates again that porting between the two systems is purely an economic decision and not a question of technical feasibility.

Recently I worked with a large application developer that had used Mirrors to port an application with over 50 separate DLLs and .EXEs. It was part of a large networked system on which dozens of users worked simultaneously. The fact that this worked at all demonstrates the similarity between the two systems. (I would like to note that Mirrors was developed in a year and a half by only five developers.)

THE SMART TOOL

Since IBM has recently announced that it will no longer develop or sell the Mirrors tool, the new alternative for porting assistance is SMART. SMART is a completely different approach to porting, but it too relies on the close similarity between the two systems for its success. Essentially, SMART performs in an automated way those tasks a team of developers would perform by hand. In a series of passes, SMART looks at the source code and helps create a new source for the target platform. That is, the conversion takes place at the source code level rather than dynamically at run time (as in WLO) or when the application is linked and bound (as with Mirrors). Ported with SMART, the application will definitely have the look and feel of a native application, and it will certainly have the speed of the new platform since it will be a native application at the end of the process.



To port an application, SMART first analyzes the source code. This analysis performs a series of measurements and look-ups based on the calls in the source application. It then either maps the call directly or inserts a number of comments into the output source code to assist the developer in making the conversion by hand. It will also provide a report that indicates the size of the porting effort.

Again, the point must be made that this is possible because the systems are similar enough that one can be mapped onto the other. Although there is not a one-to-one mapping relationship throughout Windows and OS/2, SMART takes care of the one-to-one relationships and a good deal more—even rearranging parameters when necessary. The great thing about SMART is its reporting scheme, which gives the development team a handle on the remaining effort and the marketing and financial team some idea of whether or not the effort will be worth it.

PORTING IN SUMMARY

When speaking of applications, developers and users often say things like, "It's an OS/2 app" or "This is a Windows program." This identifies the application a little too closely with the operating system. If an application does anything useful in its own right, such as managing a database or an address book, or pointing a telescope based on clicking on a star map, then this

application has a considerable amount of code that should not properly be called Windows code or OS/2 code. When you are porting an application in this context, what you are porting is the interaction with the operating system. In the case of Mirrors, people would often say that the ported application necessarily ran slower because it was using an emulation layer. But this was not necessarily true—and even when true, sometimes not significant.

For example, let's say you had a large inventory and accounting program. The actual interaction with the operating system might be an insignificant portion of the application code. It might be only a few windows, dialog boxes, and some number of menu choices. The real workhorse code would, therefore, be identical in both environments, and porting between the two would not be difficult. Nor would the ported application, even with an emulation layer, run slower because the time the CPU spends waiting between menu choices is far larger than the extra instruction cycles required for the layer. If your application is graphically intensive, this time difference does become significant and your reliance on the operating system becomes much greater. In this case, a tool like SMART becomes really useful because the result of your porting effort will be a native application that makes direct use of the graphic engine on the target system.

As Microsoft's WLO manual put it some years back, "Windows and OS/2 rely on similar concepts of dynamic linking, resource management, and windowing." That is still true today. That will still be true when Windows 95 ships. In some ways, the platforms will become even more architecturally similar since Windows will add true multitasking and threading at last. Porting between the two, in either direction, is feasible—and for many, it is also still desirable.

Derrel R. Blain is coauthor of *Real World Programming for OS/2* (Sams, 2nd Edition, 1994). He worked on the Mirrors porting tool while at Micrografx in Dallas, Tex. He is currently working at One Up Corp. as a software developer.

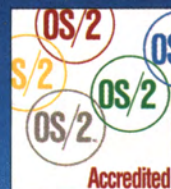
REFERENCES

Microsoft Windows Libraries for OS/2 Development Kit Programmer's Guide, Microsoft Corp.

Mirrors Toolkit User's Guide, IBM and Micrografx

Source Migration Analysis Reporting Toolset, One Up, Corp. This product is now part of the OS/2 Developer's Connection CD-ROM, which can be ordered by calling 800-6-DEVCON or (407) 982-6408.

Take OS/2 to the next level



of performance!

Classes
cover versions
2.X to Warp!



CONFERENCE & EXHIBITION

**Hynes Convention Center
Boston, MA**

Tutorials: July 17

**Conference & Exhibition:
July 18-20**

**Certification Testing:
July 18-21**

RIVETING TECHNICAL PROGRAM

OS/2 World is the only independent conference & exhibition 100% dedicated to OS/2. Choose from 120 classes that address your toughest OS/2 issues.

Our focused technical program is designed by the editors of *OS/2 Magazine* and in collaboration with IBM's Personal Software Products Division, which guarantees top-quality information. And, at OS/2 World, you'll custom design your own class schedule to get the specific OS/2 solutions you need.

As an added bonus, you'll take home a copy of the Official 1995 Conference Proceedings—so you can reference what you learn all year. Topics include:

- Client/server
- Management
- Local area networking
- System administration
- Programming
- Multimedia
- Installation and Maintenance

COMPLETE PRODUCT EXHIBITION

See and test the hottest OS/2 products. Representatives from leading companies will be ready to answer questions about your specific OS/2 challenges. You'll learn first-hand about industry trends and developments that will affect your use of OS/2 in the future.

ENGAGING SPECIAL EVENTS

OS/2 World emphasizes hands-on learning in every aspect of the conference. The wide variety of special activities offered will guarantee a good time—and a wise investment. You'll network with peers and make valuable business contacts throughout the OS/2 community. Special events include:

- OS/2 and LAN Certification Testing
- IBM Product Test Drive Area
- Three Insightful Keynote Addresses
- OS/2 User Group Meetings
- National Team OS/2 Day
- Vendor-Sponsored Evening Receptions
- Vendor Presentation Arena
- OS/2 World Bookstore



**For more info fax,
mail or call today!**



Phone: (415) 905-2354



Fax: (415) 905-2220



Mail:
OS/2 World Conf. & Exhibition
600 Harrison Street, 4th fl.
San Francisco, CA 94107



E-mail: os/2world@mfi.com



OS/2 World Conference & Exhibition

July 17-21, 1995 • Hynes Convention Center • Boston, MA

YES! I'm interested in: ☐ Attending ☐ Exhibiting

Name

Title

Company

Address

City

State/Province

Zip/Postal Code

Phone

Fax

E-mail

OSD3



Switching to flat-model programming doesn't have to be all-or-nothing. This article describes methods you can use to combine 32-bit code with 16-bit code. **By ALAN AUERBACH**

Don't Throw Away Those 16-Bit Object Libraries

If you've already made the jump (pun intended) to flat-model programming and OS/2 2.x, you know how liberating it is to be free of segment limitations. But you also know it's not always feasible to convert all your code at once to 32-bit. This is especially true when you use third-party object libraries as part of your application.

Often the process of integrating 32-bit code with 16-bit code is very straightforward. The 32-bit compiler you use is likely to have support for building thunks, which are interface routines that handle the conversion of pointers and parameters from 32-bit to 16-bit (and vice versa). Sometimes, however, you'll run into an object library that presents a special challenge. The PKWare Data Compression Library is one such library and therefore presents a good example for our discussion.

The PKWare Data Compression Library consists of two routines (`implode` and `explode`) distributed as a single Intel-format object (.OBJ) file (as well as a .LIB file). These routines, shipped as 16-bit object code, are actually operating-system independent (that is, they can be used with DOS, Windows, or OS/2). This independence is achieved because the routines call no operating system's APIs or C run-time library functions.

They use two callback routines you must declare in your code: one to read data, and one to write data.

To make use of these functions in your 32-bit program, the first thing you must do is modify the PKWare-supplied header file so the C-Set/2 compiler will supply the appropriate thunks. The original declaration of the `implode` function is shown in Figure 1, with the modified declaration in Figure 2.

The `_Far16` keyword tells the compiler that the function is 16-bit and must be called via a 16:16 pointer. The `_Pascal` keyword (which is applicable only when `_Far16` is specified) specifies the Pascal linkage conventions. The `_Seg16` keyword on the pointer declarations (in `read_buff` and `write_buff`) alert the compiler to the fact that the pointers being supplied by the calling routine are in the 16:16 format and must be converted to the 0:32 format before use.

Now we're ready to compile and link our program. We'll start with the sample program provided by PKWare: `GENERIC.C`. We've replaced the reference to `implode.h` with our modified header, `implod32.h` and changed the function definitions so they match our modified declarations.

Compiling this program results in no errors. The link step, however, shows



us the following error:

```
LINK386 : error L2050: use16/use32
attribute conflict - segment: _DATA in
group: DGROUP
```

The linker is saying here that there's a segment conflict: it's trying to combine a segment marked as `use16` (that is, a 16-bit segment) with one that's marked as `use32` (a 32-bit segment).

What's happening here? The linker has the capability to combine segments into groups. This generally improves the performance of 16-bit programs by reducing the number of segment register loads that a program must execute.

To do this, however, each segment in a group must be compatible. For example, all segments in a group must be either 32- or 16-bit. In our example, we have a 32-bit segment from our main program that the linker is trying to combine with a 16-bit segment from the `IMPLODE.OBJ` object file.

This brings us to a general rule of thumb: when combining 32-bit code with 16-bit code, put the 16-bit code in its own DLL rather than trying to statically link them together.

No problem, this just requires some slight changes to the makefile. Here we link-edit the PKWare code by itself into a DLL (we need a `.DEF` file to make it a DLL and to export the names `implode` and `explode`), build an import library to resolve the external references in our main code, and link-edit the main code referring to the import library. This time, our link step completes with no errors.

But when we run the code, we get a trap 0 at a location that resolves to `implode+72b`. When we debug this, we discover that the `implode` routine assumes that the `DS` register, on entry to the function, contains the selector of the `DGROUP` combined segment group. As it turns out, this is clearly documented in the

PKWare documentation, where it lists the segment descriptions of its data segment:

```
DGROUP GROUP _DATA
ASSUME DS: DGROUP
_DATA SEGMENT use16 WORD PUBLIC 'DATA'
```

This is not an unusual assumption for a 16-bit routine to have. After all, it is by convention that the `DS` register usually contains the selector to the `DGROUP` segment. However, it does cause us a problem here as we try to make this work from a 32-bit program.

Knowing now what we need to do (to ensure that `DS` contains the address of the group containing the PKWare data segment on entry to `implode`), we have several options: write a customized assembler-language thunk routine that explicitly loads `DS` with the correct segment selector; figure out a way to combine the PKWare data segment with our main program's `DGROUP`; or write an interface routine that does more than just act as a thunk.

I'll leave the first option as an exercise to the reader. The second option can be done with a minor change to the linker definition (`.DEF`) file. Despite the fact that the linker's error message mentions the `_DATA` segment, if we change an attribute of the `DATA32` segment (the main program's 32-bit `DATA` segment that also is grouped into `DGROUP`) to `MIXED1632`, the linker will happily combine our segments. Thus, we add the following two lines to our `.DEF` file:

```
SEGMENTS
    DATA32    CLASS 'DATA' MIXED1632
```

Now, when we link the files, we get the following warning messages:

```
implod32.obj(implode.ASM) : warning L4008:
aliased fix-up to non-alias object near 4B7
```




```
in object PK_TEXT
implod32.obj(implode.ASM) : warning
L4008: aliased fix-up to non-alias
object near 56A in object PK_TEXT
```

This is a rather important (albeit cryptic) warning, so let's decipher it one word at a time. **Aliased** is another way of saying 16:16. In a 32-bit flat model world, everything is 0:32, but if a memory object also needs to be addressed via a 16:16 pointer, that object needs an alias. A **fix-up** is a pointer that needs to be resolved at load time; this is also known as a relocation marker. **Non-alias** means 0:32, or flat-model pointer. **near 4B7** in object **PK_TEXT** is simply the location where the fix-up is found.

So what does this message mean in English? It means that a code segment (**PK_TEXT**) now contains a 16:16 pointer to a location in a 0:32 data segment. And since the first 16 bits of a 16:16 pointer contain the selector, that leaves only 16 bits to contain the offset of a location that may be more than 64K (2^{16}) bytes into the memory object. Thus, the linker warns you that you might have a problem in that case. All you have to do, though, is look back at your linker's .MAP file to make sure the segment in question lies entirely within the first 64K of the combined group. If this is true, the warning can be ignored. You can also get rid of the warning by telling the linker to add an alias

for the **DATA32** segment. You do this by adding the keyword **ALIAS** to the end of the above segment definition, so it now reads:

```
SEGMENTS
    DATA32  CLASS 'DATA' MIXED1632
    ALIAS
```

While this technique results in a working program, it can be improved by allowing the 32-bit code to not worry about segment boundaries and pass buffers larger than 64K in length. This leads us to the third option: writing an interface routine that does more than just act as a thunk.

To do this, you'll need a 16-bit compiler, so I hope you haven't

```
EXTERN unsigned short far pascal implode(
    unsigned (far pascal * read_buff)(char far *buffer,unsigned short far *size),
    void (far pascal * write_buff)(char far *buffer,unsigned short far *size),
    char far *work_buff,
    unsigned short int far *type,
    unsigned short int far *dsize);
```

Figure 1. Original declaration of the `implode` function.

```
/* typedef for read_buff function */
typedef
    unsigned short _Far16 _Pascal read_buff
        (char          * _Seg16 buffer,
         unsigned short  * _Seg16 size);

/* typedef for write_buff function */
typedef
    unsigned short _Far16 _Pascal write_buff
        (char          * _Seg16 buffer,
         unsigned short  * _Seg16 size);

/* declaration for implode function */
unsigned
    short _Far16 _Pascal implode
        (read_buff      *,
         write_buff      *,
         char            * work_buff,
         unsigned short int * type,
         unsigned short int * dsize);
```

Figure 2. Modified declaration of the `implode` function.

thrown yours away just yet. (I used the Microsoft C6.00A compiler for this article.) The idea behind this interface routine is to take as input an arbitrarily large buffer in memory, compress it, and return the compressed data to the caller. The interface code will be statically linked with the PKWare object module into a DLL.

The key to making this work is a feature of OS/2 2.0, 2.1, and

<i>Selector</i>	<i>Starting Effective Address</i>
000F	10000
0017	20000
001F	30000
0027	40000
002F	50000
...	...
FFFF	1BFF0000

Figure 3. Local Descriptor Table.

```
#define LOUSHORT(l) ((USHORT)((ULONG)l))
#define HIUSHORT(l) ((USHORT)((ULONG)l >> 16) & 0xffff))

unsigned pascal ReadBuff(char *buffer, // pointer to PKWare's buffer
                        USHORT *size) // size of buffer
{
    ULONG offset;
    USHORT len, actualLen;
    USHORT selector;

    // InputLength is a global containing the remaining length of the input buffer
    // InputBuffer is a global containing a pointer to the current position in our
    // input buffer.
    len = min(InputLength, (ULONG)(*size)); // min (remaining, buffer len)
    offset = LOUSHORT(InputBuffer); // get offset of our buffer pointer
    if (offset + len >= 65536U) // will it span 64K boundary?
    {
        actualLen = (USHORT) (65536U - offset); // get amount till boundary
        memcpy(p, inbuf32, act_len); // copy that portion to the PKWare buffer

        selector = HIUSHORT(inbuf32); // get selector
        selector >>= 3; // shift off flags
        selector++; // increment
        selector <<= 3; // shift back
        selector |= 7; // turn on flags
        InputBuffer = ((ULONG)selector) << 16; // make it a far pointer
        buffer += actualLen; // bump dest buffer ptr
        actualLen = len - actualLen; // get remaining length
        memcpy(buffer, InputBuffer, actualLen); // copy remaining stuff (if any)
        InputBuffer += actualLen; // bump pointer
    }
    else
    {
        // We won't cross a 64K boundary
        memcpy(buffer, InputBuffer, len); // copy stuff
        InputBuffer += len; // bump pointer
    }
    InputLength -= len; // subtract length
}
```

Figure 4. ReadBuff routine runs in 16-bit but is flat-model aware.

WARP known as LDT tiling. An LDT (Local Descriptor Table) is a table built by OS/2 and used by the CPU that translates a selector (the value in a segment register) to a starting "effective" address. (Actually, there is also a GDT (Global Descriptor Table). The selector contains a bit that determines which descriptor table is used: the LDT or the GDT.) LDT tiling is simply this: for each selector value in the LDT, the corresponding effective address is 64K higher than the previous value. The LDT is shown in Figure 3.

Addresses that fall within this range of tiled memory make up the OS/2 compatibility region. And, as it turns out, the compatibility regions for each release of OS/2 from 2.0 up through and including OS/2 WARP is 10000-1BFFFFFF (hex), which works out to 512 megabytes and includes the entire range of application-addressable storage.

Why not 4 gigabytes? you ask (because you know that $2^{32} = 4$ gigabytes). Because not all 16 bits of the selector are used for addressing. One bit is used to distinguish between the LDT and the GDT, and two bits are used for deter-

mining the privilege level. That leaves 13 bits of the selector that are used for addressing, plus the 16 bits of the offset, for a total of 29 bits— $2^{29} = 512$ megabytes.

Since there is no guarantee that future releases of OS/2 will provide a completely tiled address space, there are several ways you can ensure that memory you allocate will be tiled. The `DosAllocMem` API accepts a flag called `OBJ_TILE` that causes it to return tiled memory. Additionally, the C-Set++ compiler takes an option, `/Gt`, to cause all `malloc` operations to return tiled memory.

Our interface code makes use of this knowledge about tiling by examining the addresses that are passed to the `read_buff` and `write_buff` routines. If the buffer that they are handed would cross a segment boundary, then the buffer is broken up: only the portion that doesn't cross the boundary is passed first, then when the routine is called again to read (or write) more data, the next area in the next segment is passed.

To convert a 32-bit flat address (that is, a 0:32 address) into a 16-bit segment/offset address (a 16:16 address), the Compatibility Region

Mapping Algorithm is used. This is a very fancy name for a very simple algorithm. To convert from a 0:32 address to a 16:16 address, take the high-order 16 bits of the address, shift them left three bits, then arithmetically OR in the bits signifying the LDT selection and the privilege level (usually 3). Finally, the low-order 16 bits become the offset portion of the address. Thus, the flat address 00012345 becomes 000F:2345. Likewise, the conversion from 16:16 to 0:32 is the reverse process; take the selector, shift right three bits, and add the offset, as shown in Figure 4. (The LDT selection and privilege level bits are lost in this process.)

You may wonder why our 16-bit interface routines don't have a problem with the `DS / DGROUP` assumption. Ordinarily they would, but the Microsoft C6.00a compiler supports a compiler option that is specified with the memory model option. The `u` in the `/Alfu` option tells the compiler that it must explicitly load the `DS` register with its own `DGROUP` on entry to its functions. The same effect would be achieved by adding the `_loads` keyword to the definitions for the `Compress` and `DeCompress` functions.

Our interface routines also take advantage of the fact that since the `read_buff` and `write_buff` routines may get called many times throughout the course of a compress operation, they can return status information back to the caller. By passing in the address of a status monitoring function, the `Compress` and `DeCompress` routines will call that function each time the `read_buff` routine is called by PKWare.

Our Test program (Figure 5) pulls all these ideas together. It is a 32-bit multitasking Presentation Manager program that reads a file, calls the `Compress` entry point in our interface routine, and uses the status facility built in to our `Compress`

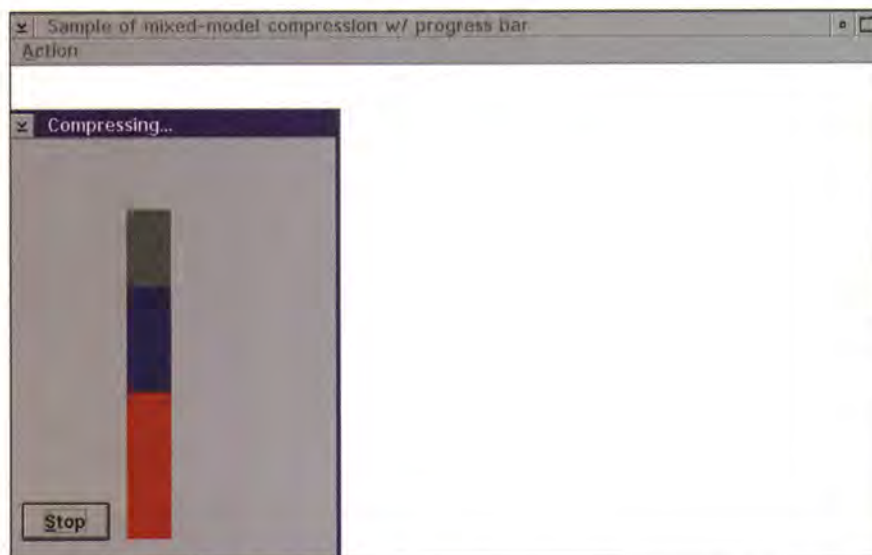


Figure 5. Our Complete Presentation Manager program with multicolored progress bar.

and DeCompress routines to put up a dialog box indicating the progress of the compression operation. The dialog box contains a vertical progress bar: the blue portion indicates that fraction of the total input buffer that has been compressed by the implode routine; the red portion indicates the size of the output buffer relative to the input buffer and gives an indication of the quality of the compression. That is, the more blue you see relative to red, the better the compression ratio. The more red you see, the worse the ratio.

In conclusion, we see that it is not necessary to throw away these nifty little object libraries just be-

cause you're writing 32-bit code. In fact, you can even enhance them by writing a little bit of interface code around them. And by understanding the concepts of segment grouping, LDT tiling, and compatibility regions, you can make your 32-bit code work seamlessly with your 16-bit code.

Alan Auerbach is an Advisory Programmer currently with the Workplace Shell Development team in Boca Raton, Fla. Prior to that, he worked in the IBM Consulting Group developing custom applications for IBM clients. He holds a B.S. in Computer Science from Rensselaer Polytechnic Institute. He can be reached at auerbach@vnet.ibm.com.

REFERENCES

At the time this article was researched, PKWare did not offer a 32-bit version of its Data Compression Library. Such a version is now available, however, and includes support for multithreaded applications.

Complete listings referenced in this article are available on CompuServe's OS2DF2 Forum, OS/2 Developer section. Download file COMPRS.ZIP.

What does it take to test a client/server system?

Synchronized, multi-user testing that goes beyond the types of testing possible with stand-alone, GUI test tools.

Unattended, extended use testing that lets you determine how your app stands up to continued pressure.

Stress, load, and performance testing to answer key questions about the activity level your system supports.

The Softbridge Automated Test Facility has been testing client/server systems since 1990. To learn how ATF can help you test OS/2, Windows, and NT apps, call 617-576-2257 (or fax 617-864-7747).



The Softbridge
Automated Test Facility

Softbridge, Inc. ▲ 125 CambridgePark Drive ▲ Cambridge MA 02140

Circle Reader Service Number 20

ASIAN OS/2 DEVELOPER SUPPORT



Get help porting OS/2 applications for Japanese, Chinese, or Korean.

Experienced guidance from the leading double byte character set OS/2 developer. Training or consulting on either an hourly or long term basis.



MicroBurst, Inc.

9035 Shady Grove Court
Gaithersburg, MD 20877

(301) 330-2995

Fax: (301) 330-8609

CompuServe: 70334,3616

Internet: 70334.3616 @
COMPUSERVE.COM.

Circle Reader Service Number 21



Send text messages to pagers directly from OS/2!



- Easy-to-use workplace shell application
- Includes a command line interface
- You can 'page-enable' your applications!
- 32-bit, SOM-based code - Only \$79

ChipChat® Wireless Communicator

ChipChat-Cawthon Software
Dearborn, Michigan USA & Fukuoka Japan
Phone 313-565-4000 Fax 313-565-4001

Circle Reader Service Number 25

FAX Developer Tools

From the developers of *FaxWorks for OS/2*

Client/Server API and Printer Driver Toolkits

Support for LANs, up to 96 lines per CPU,
and all popular fax hardware

Keller Group Inc.

Voice: (612) 429-7273

Fax: (800) 329-3293

Email: kgroup@ibm.net

Fax: (612) 653-1987

Faxworks is a trademark of Global Village Communication.

Circle Reader Service Number 26

TCP/2™

TCP/IP for OS/2®

Supports *all* released versions of OS/2
including 32bit API for OS/2 2.0 and above

**Our customers
say our tcp/ip is the best.
It's faster, more reliable,
and more complete.
Find out for yourself!**

A DAN LANCIANI PRODUCT
For further information contact: Essex Systems, Inc.
One Central Street • Middleton, MA 01949

(508) 750-6200

TCP/2 is a trademark of DLD consulting. Ethernet is a trademark of Xerox Corporation.
OS/2 is a registered trademark of International Business Machines Corporation. VT102 is
registered trademark of Digital Equipment Corporation. TCP/2 is based in part on
work done by the University of California Berkeley.

Circle Reader Service Number 28

JOB SCHEDULING SERVER

Presentation Manager 32-bit Job Scheduler for NOVELL,
IBM LANS, and stand alone machines. Schedules multiple
OS/2, DOS, and Windows jobs for concurrent execution
on specified or auto selected LAN nodes. Conditional job
scheduling, job logging, sophisticated calendar, security,
priority, job recovery, user APIs, and much more.

Microwork, Inc.

Phone: (708) 940-8979

Fax: (708) 940-8979

Data Acquisition for OS/2

Presenting IOPRO/VX. This toolkit allows you to unleash the 32
bit multitasking power of OS/2 and the incredibly easy to use
development environment of VX-REXX against any and all of
your data acquisition or process monitoring and control applica-
tions! Plug in your favorite I/O adapter, drag and drop to build
your app, and that's it. Or, create your own screen instrument
panels and remotely control your instrument via RS-232, IEEE-
488, or a network!

Call or write for a free brochure.

"If it's done right, chances are it's Rock Solid."

Rock Solid Software
8460 Plank Road Box 228
Montville, OH 44064
(216) 390-0590

OS/2 is a registered trademark of International Business Machines Corporation.
VX-REXX is a trademark of Watcom International Corp.

Circle Reader Service Number 29

All mallocs Are NOT Equal

SmartHeap™ is an ANSI portable malloc and new for OS/2, Windows, NT,
Macintosh and UNIX. It delivers speed improvements up to 100X versus
compiler-supplied libraries, especially for large heaps in the presence of virtual
memory. In addition to malloc and new, SmartHeap provides a fast fixed
size allocator, supports multiple pools, and includes a debugging library which
catches leaks, overwrites, invalid references, and other heap-related errors.
Exceptionally reliable - used by Sybase, Oracle, Candle, Legend, Adobe,
Informix, Autodesk, WordPerfect, HP, Borland and many others.

Call for Free White Paper, Benchmarks, and Error Reports

No
royalties

MicroQuill

60 Day Money-
Back Guarantee

800-441-7822

206-525-8218 / Fax 206-525-8309 / Compuserve: 70751,2443 / Internet: info@microquill.com

Circle Reader Service Number 30

SUBSCRIBE TODAY!

Only \$39.95 for a full year subscrip-
tion to the premier source of tools and
techniques for the OS/2 software
developer: **OS/2 DEVELOPER**

Call today and don't miss a single issue:

1-800-WANT-OS2

The easy to use 32 bit OS/2 PM IPF Language Editor...

IPF Editor

- Add help to applications in minutes
- Designed for easy use by programmers and non-programmers
- Generate C and Resource source files to add help to applications automatically
- Import wordprocessing files, including WordPerfect, quickly and accurately
- Preview IPF output without compiling
- Create all hypertext links automatically
- Optional Voice Support Module
- Includes complete on-line help and documentation

IPF Editor: \$150
Voice Support Module: \$50

Orders:
1-800-IPF-7622

Information: (360)428-5025
on compuserve at
70410,2416

Perez Computing Services
4725 Monte Vista Place
Mt Vernon, WA 98273

Circle Reader Service Number 33

Tasker 3.0 Task Scheduler for OS/2 From Evolutionary Software

- Implemented As A Workplace Shell Object
- Start DOS, Windows, OS/2 Programs
 - All DOS Settings Available
 - Graphical Holiday Calendars
- Automatic Logging of Critical Events
- Programs Available to Add User Entries To The Log File
 - Exception Days and Dates
 - Graphical Log Viewer
- Repeat Intervals Include: Minutes, Hours, Days, Weeks, Months Workdays, Existence Or Nonexistence of a File

Sales (219) 833-4556

1 Copy \$79.95 5-10 Copies \$69.95

Circle Reader Service Number 34

VisualAge & PARTS

Spreadsheets, tables, and business graphics are easy to add to your VisualAge™ and PARTS™ applications.

WidgetKit™/Professional provides spreadsheets, multi-column list boxes, table editor, bitmap viewers, input validation, virtual spreadsheets, printing, and more.

WidgetKit/Business Graphics has more than 50 graphs and charts types, 2 and 3-D. Control color, fonts, printing, and more.

No royalties for end-user applications, 90 days free support, and 30-day money-back guarantee. Call for FREE info.

Objectshare Systems, Inc. v: 408.970.7280 f: 408.970.7282
5 Town & Country Village, Suite 735, San Jose, CA 95128

Circle Reader Service Number 35

Opt-Tech Sort/Merge

High Performance Sort/Merge/Select Utility

Use as a stand-alone utility or call as a Subroutine. Many features including unlimited file size, multiple keys, record selection, duplicate elimination, summing and more!

Call for free brochure Opt-TechData Processing
P.O. Box 678

Available for: Zephyr Cove, NV 89448
OS/2 or Unix \$249 Phone (702) 588-3737
DOS or Windows \$149 Fax (702) 588-7576

Circle Reader Service Number 32



What Are You Waiting For?

Right now you could be
reaching 36,000+ product-buying
OS/2 Developers!

You can run an 1/8 page Products & Services Guide ad
for as little as \$350/insertion...

...a double-sized ad for as little as \$600/insertion!!!

The May/June issue of OS/2 Developer focuses on GUI
development

Space close date is March 2

CALL NOW FOR DETAILS

KRISTIN MORGAN
V/212/626-2498 F/212-869-0899
E/kmorgan@mfi.com

May/June issue theme:
GUI DEVELOPMENT
AD CLOSE DATE: MARCH 2

Invest a stamp



Save a bundle

For the price of a stamp, you can get the latest edition of the federal government's free **Consumer Information Catalog** listing more than 200 free or low-cost government publications on topics such as federal benefits, jobs, health, housing, education, cars, and much more. Our booklets will help you save money, make money, and spend it a little more wisely.

So stamp out ignorance, and write today for the latest free **Catalog**. Send your name and address to:

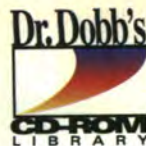
**Consumer Information Center
Department SB
Pueblo, Colorado 81009**

A public service of this publication and the Consumer Information Center of the U.S. General Services Administration.

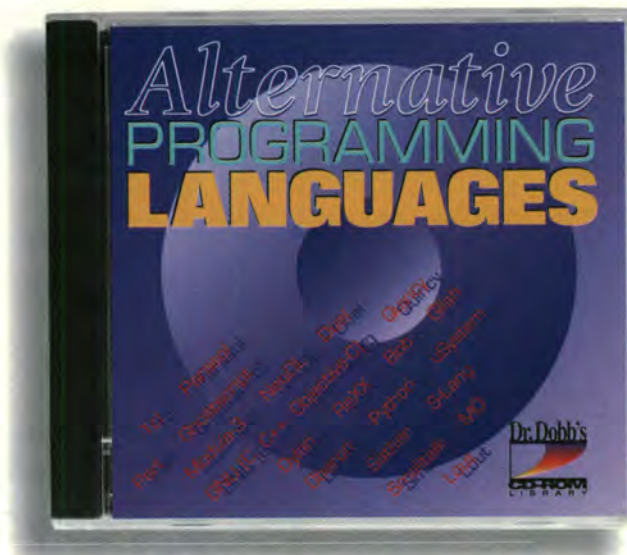
ADVERTISER INDEX

Reader Service	Company Name	Page
13	Athena Design	35
25	ChipChat	62
8	Compuware	17
11	Development Technologies	31
28	Essex Systems	62
34	Evolutionary Software.....	63
3	Hockware	COV4
5	IBM.....	6-7
16	Indelible Blue.....	37
—	Inmark	1
14	JBA.....	36
26	Keller Group	62
18	MHR.....	50
1	Metaware.....	COV2
21	Micro Burst.....	61
19	MicroEdge.....	51
30	MicroQuill.....	62
—	Microwork.....	62
12	Mitnor	34
35	Objectshare.....	63
32	Opt-Tech.....	63
36	One-Up	23
—	OS/2 World	54-55
33	Perez.....	63
15	Pinnacle	46
4	Programmer's Paradise	4
6	Prominare.....	9
—	Quadron	73
23	Rimstar.....	COV3
29	Rock Solid.....	62
20	Softbridge	61
7	Stac	15
37	Suitable Alternatives	79
17	Tech Bridge	47
10	Veritas	25
9	Watcom.....	2
38	Zinc.....	21

Alternative Programming Languages CD-ROM



This CD-ROM is your definitive resource for the latest in cutting-edge programming environments. Gain access to the most innovative, solution-providing languages and tools available today!



Examine the next generation of languages

You need this tool to get up to speed on distributed computing, embeddable languages, increased productivity, and languages of the future.

You get all these languages, compilers, interpreters - Perl, Glish, Smalltalk, Sather, Tcl, M0, Modula-3, ReXX, Lout, Ghostscript, Dylan, Duel, Bob, Neudl, Python, Oberon, Parasol, S-Lang, Quincy, uSystem, Distributed C, the GNU compiler suite—and more!

Put this CD-ROM to work immediately on your development projects:

- **Object-Oriented Development.** Tired of C++? Look no further than Sather, Dylan or Bob and get re-energized.
- **Distributed Computing.** When building distributed systems, languages like Parasol and M0 make your life easier.
- **Multiple Platforms.** Check out Perl and Python, plus a wide range of compilers, including the GNU compiler suite.
- **Text Processing and Document Formatting.** Lout and Ghostscript let you edit and produce complex documents with ease.
- **Prototyping.** When prototyping connected distributed systems, Glish makes it all happen for you.
- **Neural Networks.** Neudl takes the pain out of building complex neural networks.
- **And much more!**

FREE CALL!
800-431-8567
24 HOURS A DAY
7 DAYS A WEEK

All these languages are robust, imaginatively designed, break new technical ground, and come with source code for compilers or interpreters. And you also get the full descriptions and specifications.

SPECIAL BONUS!

With your purchase of this CD-ROM, you'll also receive a copy of the *Dr. Dobb's Sourcebook, Alternative Programming Languages* A \$4.95 value — ABSOLUTELY FREE!

YES! Send me the Dr. Dobb's Alternative Programming Languages CD-ROM immediately. I will also receive the *Dr. Dobb's Sourcebook Alternative Programming Languages* as a bonus.

My price is \$49.95 (plus shipping: \$2.00 U.S./Canada; all other countries \$12.50). California residents, please add \$4.25 sales tax. **Clip out or photocopy this form.**

Name _____

Address _____

City/State/Zip _____

Country _____

Here's how I'm paying (circle one): VISA - MC - AMEX - Check enclosed

Credit card number (include all digits) _____

Expiration Date _____

Signature required _____

OS/21

For more information, send E-Mail to:

Alternative Programming Languages CD
info_altcd@mfi.com

FREE CALL: 800-727-3662 24 HOURS A DAY, 7 DAYS A WEEK **FAX ORDERS: 510-372-8582**
E-MAIL ORDERS: sbarnes@mfi.com **MAIL ORDERS:** ALT CD, P.O. Box 1525, Martinez, CA 94553
INTERNATIONAL: USE MAIL, FAX 510-372-8582, E-MAIL sbarnes@mfi.com, OR CALL 415-655-4190



This article shows Windows programmers how to ease the transition to OS/2 and includes a useful list of Dos and Don'ts. By **LOU MIRANDA**

OS/2 Programming for the Windows Guru

O.K., so you're a real Windows programming guru: you've learned all the ins and outs and know all the tips and tricks. But now your organization wants you to learn OS/2 programming, perhaps because of the success of OS/2 Warp or the portability to the PowerPC that it offers. You're starting to feel ill at ease. It's similar to the transition from high school to college: you're going from top dog to neophyte in one swift jump.

It's really not all that bad. In fact, if you've been in the PC industry for a few years, you may recall the transition you had from DOS to Windows—that was very tough indeed. The problem with the DOS to Windows transition was that you went from a text-based, procedural environment (DOS) to a GUI event-driven interface (Windows).

Luckily for you, the transition from Windows to OS/2 programming is much, much easier. While OS/2 certainly has a number of advanced features that Windows doesn't—like threads and semaphores—the basic programming paradigm hasn't changed. And if you started learning the Win32 API, which does share many features with OS/2, you are well on your way to being an OS/2 programming guru.

THE EASY WAY OUT

There are some simple ways of easing the burden: cross-API libraries and code converters. An example of the latter is SMART (Source Migration Analysis Reporting Toolset) from One Up Corp. This is a useful tool that now comes free on the Developer Connection CD-ROM. It reads your source code and converts most of it to native OS/2 calls. What it can't convert, it gives you advice on converting yourself. Of course, the more you know about the OS/2 API, the more helpful it will be.

A second easy option is a cross-platform library, such as Zinc, ZApp, Borland's Object Windows Library (OWL) for Presentation Manager (PM), and XVT. These C and C++ frameworks let you port the majority of your code by just recompiling. Since they don't support 100% of OS/2's features, it helps to know the OS/2 API so you can extend the framework in the few areas not supported.

STRIKING SIMILARITIES

Windows and OS/2's PM follow the same basic programming model: a message-based, event-driven programming interface. Figure 1a shows a minimal Windows program and Figure 1b a minimal OS/2 PM program. While some of the function names have changed, and



OS/2 adds an extra call or two, there is clearly a dramatic similarity between the two.

This similarity extends to files associated with the source code. In Windows, you `#include <Windows.h>` in your code to get all the Windows macros, `#defines`, and function prototypes. Similarly, in OS/2 source code, you `#include <OS2.h>` to get all those requirements. However, there's a catch: before including that file, you have to `#define` some items so that it includes the right items. For example, any windowing code has to `#define INCL_PMWIN` before it can `#include OS2.h`; likewise you need to `#define INCL_PMGPI` for graphics and `INCL_DOSPROCESS` for threads. The online help for each function lists the `INCL_...` `#defines` that you must use.

OS/2 and Windows are also similar in that they both have resource (*.RC) files that contain resources such as menus, bitmaps, icons, string tables, and

dialog boxes. Again, OS/2 adds a few refinements (such as help tables and association tables), but the similarity is striking (and calming!). This is no surprise, of course: Microsoft and IBM worked together on OS/2 version 1.x.

THE OS/2 API

Naming Conventions. Lest you get the impression that converting Windows code to OS/2 code is trivial, let's jump into a discussion of the differences. Look closely at Figures 1a and 1b again. You will notice that while the function names are very similar in both listings, all the OS/2 functions begin with `Win`. For example, where Windows has `GetMessage()` and `CreateWindow()`, OS/2 has `WinGetMsg()` and `WinCreateWindow()`.

This naming convention of prefixing a three-letter code to the function name makes a lot of sense. It presents a nice, cohesive organization to the programmer. Table 1 gives a listing of the three-

<i>Operating System Functionality</i>	<i>Three-letter Abbreviation</i>	<i>OS/2 API Example Functions</i>	<i>Windows Equivalent Functions</i>
Windowing	Win...	WinGetMsg WinCreateWindow	GetMessage CreateWindow
Graphics Programing Interface	Gpi...	GpiCharString GpiBox	TextOut Rectangle
Profile (INI) file	Prf...	PrfQueryProfileInt PrfWriteProfileString	GetPrivateProfileInt WritePrivateProfileString
Drag-and-Drop	Drg...	DrgAcceptDroppedFiles DrgQueryDragItem	DragAcceptFiles DragQueryFile
Disk Operating System	Dos...	DosLoadModule DosQueryModuleName	LoadModule GetModuleFileName
Device	Dev...	DevQueryCaps	DeviceCapabilities

Table 1. OS/2's API is divided into sections based on a three-letter prefix.


```

// Windows Hello for OS/2 Developer: Bare-bones Hello program for MS Windows

#include <windows.h>           // include the main Windows header file

// declare your window procedure
LRESULT PASCAL WndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

// declare your main procedure
int PASCAL WinMain(HINSTANCE hCurInstance, HINSTANCE hPrevInstance,
                  LPSTR /* lpCmdLine */, int nCmdShow)
{
    static char appName[] = "HelloWin";
    HWND hWnd;
    MSG msg;
    WNDCLASS wndclass;

    if (!hPrevInstance)
    {
        // create & register your window class, with the desired attributes
        wndclass.style      = CS_HREDRAW | CS_VREDRAW;
        wndclass.lpfnWndProc = WndProc;
        wndclass.cbClsExtra  = 0;
        wndclass.cbWndExtra  = 0;
        wndclass.hInstance  = hCurInstance;
        wndclass.hIcon      = LoadIcon (NULL, IDI_APPLICATION);
        wndclass.hCursor    = LoadCursor (NULL, IDC_ARROW);
        wndclass.hbrBackground = GetStockObject (WHITE_BRUSH);
        wndclass.lpszMenuName = NULL;
        wndclass.lpszClassName = appName;

        RegisterClass (&wndclass);
    }

    // create the main window
    hWnd = CreateWindow (appName,

                        "Hello, OS/2 Developer",
                        WS_OVERLAPPEDWINDOW,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        CW_USEDEFAULT,
                        NULL,
                        NULL,
                        hCurInstance,
                        NULL);

    // display the main window
    ShowWindow (hWnd, nCmdShow);
    UpdateWindow (hWnd);

    // start processing the messages sent to you in the message queue
    while (GetMessage (&msg, NULL, 0, 0))
    {
        TranslateMessage (&msg);
    }
}

```

Figure 1a. Sample Windows program (continued on page 69).



```
        DispatchMessage (&msg);
    }
    return (msg.wParam);
}

// your window procedure for the main window
LRESULT PASCAL WndProc (HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam)
{
    HDC hDC;
    PAINTSTRUCT paintStruct;
    RECT rect;

    switch (uMsg)
    {
        // paint (draw) the main window
        case WM_PAINT:
            hDC = BeginPaint (hWnd, &paintStruct);
            GetClientRect (hWnd, &rect);
            DrawText (hDC, "Windows Hello for OS/2 Developer!", -1, &rect,
                    DT_SINGLELINE | DT_CENTER | DT_VCENTER);
            EndPaint (hWnd, &paintStruct);
            return (0);

            // exit the application
        case WM_DESTROY:
            PostQuitMessage (0);
            return (0);
    }

    // if you don't process the message, let Windows take care of it
    return DefWindowProc (hWnd, uMsg, wParam, lParam);
}
```

Figure 1a. Sample Windows program (continued from page 68).

letter codes, what they stand for, and a few examples of each and their Windows equivalents.

Careful examination of Table 1 reveals other subtle changes. For example, many `Get...()` functions in Windows become `[Win]Query...()` functions in OS/2. Also, there has been consolidation in some functions. For example, `GetParent()` in Windows becomes `WinQueryWindow (QW_PARENT)` in OS/2. Of course, OS/2 also has a number of API features that Windows 3.1 doesn't, including Workplace Shell, spooler, and threads. Some of these have equivalents in Windows95.

Messages, Parameters, and Return Values. Message names are very

similar in Windows and OS/2. In fact, many are identical. `WM_COMMAND`, `WM_CHAR`, and `WM_PAINT`, for example, are present in both. Minor differences do crop up, of course. Windows has `WM_LBUTTONDOWN` and `WM_RBUTTONDOWNBLCLK`, where OS/2 has `WM_BUTTON1DOWN` and `WM_BUTTON2DBLCLK`.

Both Windows and OS/2 messages have the same number of parameters (two), but their types are different:

- Windows parameters: `WPARAM wParam`, `LPARAM lParam`
- OS/2 parameters: `MPARAM mp1`, `MPARAM mp2`.

`WPARAM` is a word (16 bit), and `LPARAM` is a long (32 bit), while both `MPARAM`'s are longs (32 bit). Since the

types are different, and because OS/2 has a more consistent ordering of the parameters, messages with the same name in both Windows and OS/2 often have parameters in different orders or with completely different contents. For example, let's look at the `WM_CHAR` message. Compare the parameters:

- Windows: `wparam` is a short and `lparam` is a long
- OS/2: `mp1` comprises a short and two chars; `mp2` comprises two shorts.

So the moral of the story is that just because messages share a name doesn't mean you can interchange their parameters.


```

// HelloOS2.Cpp, a Hello application for OS/2 Developer
// From an article by Lou Miranda

// define the section(s) you want to include from os2.h
#define INCL_WIN
// the main OS/2 header file
#include <os2.h>

int main (void);
// use extern"C" if you're using C++
extern "C" {
MRESULT EXPENTRY ClientWndProc( HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2);
}

int main( void)
{
    HAB hab;          // an anchor block; used only for retrieving error information
    HMQ hmq;          // a message queue--where you get your application's messages
    QMSG qmsg;
    HWND hwndFrame, // the frame window (equivalent to the Windows "non-client"
                    hwndClient; // the client window--where you do your drawing
    // set your frame window's flags; note that we remove the menu, icon, and
    // accelerator table flags, because if they are included then these
    // resources **must** be defined in the RC file, which we don't have
    ULONG flFrameFlags = FCF_STANDARD & ~FCF_MENU & ~FCF_ICON & ~FCF_ACCELTABLE ;
    CHAR szClassName[] = "HelloOS2" ;
    CHAR szWindowTitle[] = "Hello, OS/2 Developer" ;

    hab = WinInitialize (0);          // initialize OS/2's windowing
    hmq = WinCreateMsgQueue (hab, 0L); // create a default-sized message queue

    // register your window class for the client window
    WinRegisterClass (hab, szClassName, ClientWndProc, CS_SIZEREDRAW, 0);

    // create a "standard" window, which is a frame window plus your client
    hwndFrame = WinCreateStdWindow (HWND_DESKTOP, 0L, &flFrameFlags, szClassName,
                                    szWindowTitle, 0L, NULLHANDLE, 0L, &hwndClient);

    // display the window
    WinSetWindowPos (hwndFrame, HWND_TOP, 0, 0, 0, 0,
                    SWP_SHOW | SWP_ACTIVATE | SWP_ZORDER);

    // start processing the messages for your main window
    while (WinGetMsg (hab, &qmsg, NULLHANDLE, 0L, 0L))
        WinDispatchMsg (hab, &qmsg);

    WinDestroyWindow (hwndFrame); // destroy your main window
    WinDestroyMsgQueue (hmq);     // destroy the message queue
    WinTerminate (hab);           // terminate the use of OS/2's windowing

    return 0L;
}

// your client window procedure
MRESULT EXPENTRY ClientWndProc (HWND hwnd, ULONG msg, MPARAM mp1, MPARAM mp2)

```

Figure 1b. Sample OS/2 program (continued on page 71).


```

{
    HPS        hps;    // a handle to a Presentation Space--what you draw into
    RECTL      rectL;  // the drawing rectangle

    switch (msg)
    {
        // paint your client window
        case WM_PAINT:
            hps = WinBeginPaint (hwnd, NULLHANDLE, &rectL);
            WinQueryWindowRect (hwnd, &rectL);
            WinFillRect (hps, &rectL, SYSCLR_WINDOW);
            WinDrawText (hps, -1, "OS/2 Hello for OS/2 Developer!", &rectL,
                SYSCLR_WINDOWTEXT, SYSCLR_WINDOW, DT_CENTER | DT_VCENTER);
            WinEndPaint (hps);
            return (MRESULT) 0;

        default:
            break;
    }
    // let OS/2 handle any message you didn't process
    return WinDefWindowProc (hwnd, msg, mp1, mp2);
}

```

Figure 1b. Sample OS/2 program (continued from page 70).

Sometimes, as in the WM_CHAR example, the MPARAMs contain values other than longs. How do you retrieve the values contained within them? The OS2.H file contains a number of macros to assist you. To retrieve the short value from WM_CHAR's mp1, use SHORT1FROMMP(mp1). To retrieve the two chars, use CHAR3FROMMP(mp1) and CHAR4FROMMP(mp1). To retrieve the two shorts from mp2, use SHORT1FROMMP(mp2) and SHORT2FROMMP(mp2).

Of course, sometimes you need to pack data into a parameter, rather than extract them out of it. There are macros for that too. To create an MPARAM composed of two shorts, use MPFROM2SHORT(x,y), where x and y are short variables. Another variation on that theme is MPFROMH-WND(hwnd). There are a variety of similar macros; search the online help for "Helper Macros" for a list of all of them.

Message return values are different in OS/2, compared with Windows. In Windows, messages

return an LRESULT, while in OS/2, they return an MRESULT type. Just like the message packing and unpacking macros, there are macros for packing and unpacking message results—for example, MRFROM2SHORT(x,y) and MRFROMLONG(1).

WINDOWING AND DRAWING DIFFERENCES

Now that you've got a good feel for the API changes, let's discuss some of the changes you'll discover as you write your first application.

Your main application window

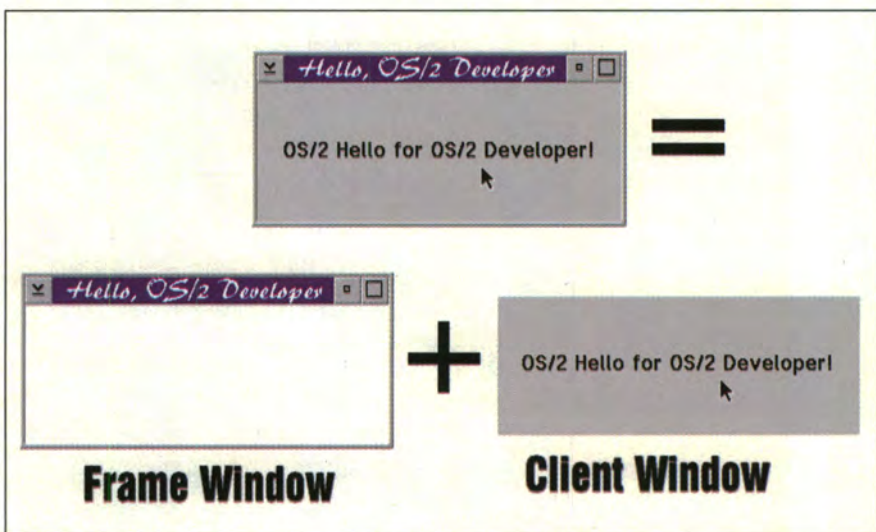


Figure 2. Standard OS/2 windows have both a frame window and a client window. The frame window includes the resizable frame, system menu/titlebar icon, titlebar, and minimize/maximize buttons. The client window is where you usually do your drawing.



is actually composed of two windows in OS/2: the outer frame window and the inner client window. A similar distinction exists in Windows, but the outer area is called the nonclient or NC area of your application window. For example, to track the mouse in the NC area in a Windows window, you check for the `WM_NCHITTEST` message. So when you want to draw in your application, you'll usually want to draw in the client window, not the frame window, as illustrated in Figure 2.

When you do get ready to draw, OS/2's default mapping treats the graphics point (0,0) as the lower left corner of your window; in Windows, (0,0) is the upper left corner.

Windows abstracts hardware

devices with the concept of the HDC, or handle to a device context. OS/2 uses the same abstraction, but it adds another level, the HPS, or handle to a presentation space. Most of the Windows graphics device interface (GDI) calls use the HDC directly. In OS/2, the graphics programming interface (GPI) calls use the HPS instead.

The GPI in OS/2 is much richer than the GDI of Windows 3.1 and Windows 95. OS/2's GPI offers transformations, so you can rotate, skew, enlarge, reduce, and offset the drawing of lines, arcs, circles, rectangles, and text. This is a powerful feature that drawing and graphing programs will want to take advantage of. Another powerful concept absent in

Windows is that of a bundle. Graphics bundles are similar to styles in a word processor: they let you predefine a set of attributes you can then use when drawing different parts of your window. OS/2 has bundles for characters, lines, and areas.

DOS AND DON'TS

By now you're pretty hip to programming in OS/2. You've got the basics down pat. So let me toss you some hints that will make your life easier as a programmer or make your users happier.

Don't:

- Don't be tempted to use OS/2's memory allocation features (such as `DosAllocMem()`). Although they can be very useful in specific situations (such as omitting the `COMMIT` flag for creating sparse arrays in memory), your code will be much more portable if you simply use the C/C++ run-time memory allocation functions, such as `malloc()` and `new()`. Also, be aware that using `DosAllocMem()` allocates memory in 4K blocks, even if you request only 1 byte! Again, using the C/C++ functions will prevent this.
- Don't use OS/2's file routines (such as `DosOpen()` and `DosRead()`), unless you need OS/2-specific features such as extended attributes. Using the C/C++ file functions will result in more portable code.
- Don't use `DosCreateThread()` if you will be using any C/C++ run-time functions in your thread. Use `_beginthread()` instead since it properly initializes the C/C++ run-time environment.

Do:

- Do use OS/2's CUA '91 controls, such as the notebook, container, value sets, proportional scroll bars, and the spin control. Keeping your application similar to other programs will make

Control Messages

There are significant changes regarding control notification messages. In Windows, when a control notifies its parent (for example, when the user double-clicks in a listbox), the application gets a `WM_COMMAND` message. In OS/2, the application gets a `WM_CONTROL` message instead.

Let's compare what happens in OS/2 and Windows when a user double-clicks in a listbox control. In Windows, the application gets a `WM_COMMAND` message, where `wparam` contains the listbox ID and `lparam` contains the listbox's `HWND` and the `LBN_DBLCLK` notification message. In OS/2, on the other hand, the application gets a `WM_CONTROL` message, where `mp1` contains the listbox ID and the `LN_ENTER` notification message, and `mp2` contains the listbox's `HWND`. So while the kinds of information passed are basically the same, the ordering of the data and names of the messages change. It's the little things that'll get you!

There is another thing to watch for in converting your code for controls: OS/2 consolidates the names of messages for "composite" controls. For example, a combo box is a composite of an entry field and a listbox. Windows treats this as a completely unique control, so if a user double-clicks in it, the application gets a `CBN_DBLCLK` (combo box double-click) message. In OS/2, the application simply gets an `LN_ENTER` (listbox enter) message. This is a simpler, more consistent way of doing it, but it is something you must be aware of. So don't go looking for a `CN_ENTER` message, because it doesn't exist.



users feel more comfortable, and the environment will be more consistent.

- Do use threads where appropriate. Nothing is more annoying to a user than having a powerful system like OS/2 that's paralyzed by a single-threading program. This is especially important in PM applications. The rule of thumb is that if it is going to take more than 0.1 second to process a message, send the processing off to a separate thread so that PM doesn't get locked up.
- Do try to integrate your application into the Workplace Shell, even if it isn't SOM-enabled. For example, make sure your application takes advantage of the drag-and-drop API. Make sure it uses the ASSOCTABLE resource in your *.RC file so OS/2 creates a default template for your application's data files. Also make sure it takes advantage of the right mouse button for menus (by trapping the WM_CONTEXTMENU message, not WM_BUTTON2UP).

CONCLUSION

By now you should have a pretty good feel for OS/2 programming. While this article certainly lists a good number of differences, it's important to realize that these are, for the most part, small changes; the core programming paradigm hasn't changed. Once you know Windows programming well, moving to OS/2 is an easy transition. Just remember to incorporate OS/2-specific features whenever you can. Your customers will thank you.

Lou Miranda is a former Contributing Editor to PC WORLD magazine. He has a master's degree in molecular biology; is currently a St. Paul, Minn.-based OS/2 programming consultant; and is working on a book on OS/2 for the PowerPC. Lou can be reached on CompuServe at 73567,471 or via e-mail at lmirand@ibm.net.

Quadron tools cut 32% off the average development time.

Take it from our customers. A recent survey determined that, by using Quadron communications development tools, they cut 32% off their estimated development time.

Increase your efficiency

Our development software allows you to shift communications tasks from the CPU on a PC to a specialized IBM ARTIC co-processor card that's designed to handle them. You'll gain CPU independence and multiple high-

Tools that get the job done

We have off-the-shelf solutions for your common data transmission needs, and tools for custom situations. Select from HDLC/SDLC, Bisync, Async, X.25, LAP-B and custom protocols.

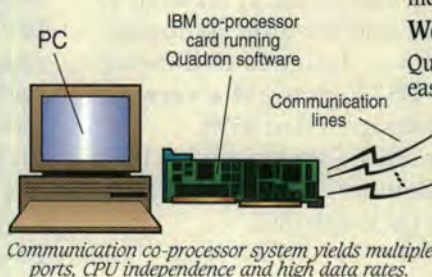
Your PC can be an ISA, EISA or MCA machine on which you can run various operating systems. Perhaps best of all, you can port the co-processor code you write from one operating system to another with little or no modification.

Work faster & simpler

Quadron software is easy to install, easy to use, and downright productive. Our high-level, C-language API makes your co-processor programming faster. The user manuals have numerous code examples to speed you on your way. And our support, should you need it, will serve up fast and accurate answers to your questions, whether they come in by phone, fax, or through our BBS.

Act right away

Does this sound like something you could benefit from? Contact us right now.



speed ports. Each port on the IBM card can potentially support a different protocol and a different line speed.

Applications everywhere

Today, Quadron helps people manage demanding applications like:

- Travel reservation
- Stock market data delivery
- Telecommunications switching
- Retail point-of-sale purchases
- Process control
- Automatic teller services
- Shop floor control.



Quadron

209 East Victoria Street
Santa Barbara, CA 93101 USA
fax 805-966-7630
telephone 805-966-6424



© 1995 Quadron Service Corporation. All trademarks are the property of their owners.



Buyer's Guide

The staff of OS/2 Developer put together this special section to guide you through the various software tools specifically for application migration. The products described in this guide are based on surveys sent to vendors; they are provided as a service to you and are free to vendors. These listings do not represent an endorsement by OS/2 Developer. Although every effort has been made to ensure accuracy, we do not assume any responsibility for error or omission in this section.

Application Migration Buyer's Guide

AUTUMN HILL SOFTWARE INC.

Circle No. 101

Menuet/CX 4.0 is a multiplatform GUI application framework for C++. Features include CUA compliancy, a range of GUI controls, and prebuilt dialogues. Menuet/CX was designed and developed under OS/2, giving it a versatile architectural design. Price: \$599.

Autumn Hill Software Inc., 1145 Ithaca Dr., Boulder, Colo. 80303, (303) 495-8865, fax (303) 494-7802.

FLEXUS INTERNATIONAL CORP.

Circle No. 102

COBOL spII 3.1 is a screen management tool for COBOL programmers. It is designed to deliver transparent migration of the user's application screens and source code across text and GUI interfaces. Benefits include cross hardware and operating system portability. It provides an interactive screen painter, COBOL code generator, and functional run-time processing. It allows users to develop screens once and run them in a variety of environments. COBOL spII sorts screen definitions in memory-resident COBOL and maintains one COBOL source program with an unlimited number of screen file options available to customers or end users. Price: \$1,000.

Flexus International Corp., P.O. Box 640, Bangor, Pa. 18013, (610) 588-9400, fax (610) 588-9475.

INMARK DEVELOPMENT CORP.

Circle No. 103

zApp Developer's Suite 2.2 solves the problem of writing commercial quality C++ applications that run on multiple platforms. It consists of zApp Application Framework, zApp Interface Pack, and zApp Factory. Applications generated using the zApp Developer's Suite are single-source portable to OS/1, Windows, Win32, DOS Text, DOS Graphics, and X/Motif. Price: \$1,295.

Inmark Development Corp., 2065 Landings Dr., Mountain View, Calif. 94043, (800) 346-6275 or (415) 691-9000, fax (415) 691-9099.

JBA INTERNATIONAL

Circle No. 104

Guidelines Desktop 2.1A is an OS/2-hosted, graphical, event-driven application development tool that generates C++ code. It provides over 30 drag-and-drop Presentation Manager objects. It utilizes an object-oriented language, JOT with full point and click prompting to aid programmers. Guidelines Desktop is fully SOM-enabled. Supported clients are OS/2 and Windows. Servers include the DB/2, DB2/400, DB2/6000, Oracle,



and ODBC. Price: \$595 to \$2,380 depending on components.

JBA International, 3701 Algonquin Rd. Ste. 1000, Rolling Meadows, Ill. 60008, (800) 522-4685, fax (708) 590-0394.

LIANT SOFTWARE CORP. Circle No. 105

C++/Views is an object-oriented application framework for developing native GUI programs that run on Windows, Windows NT, OS/2, Macintosh, or OSF/Motif. C++/Views includes a library of C++ classes for developing GUI applications. It includes C++/Views Constructor, a unique visual development tool that unites an interface builder with a class browser. Price: \$1,499.

Liant Software Corp., 959 Concord St., Framingham, Mass. 01701, (800) 237-1873 or (508) 875-2246, fax (508) 820-0035.

NEURON DATA INC. Circle No. 106

Elements Environment is a suite of products that provide an environment that enables development of high-end client/server applications. The GUI builder component includes a comprehensive set of graphical objects. The data access component lets you tap into a range of data sources across multiple platforms and separates the logical view from the physical data source. Price: starts at \$6,850. There are no run-time fees.

Neuron Data Inc., 156 University Ave., Palo Alto, Calif. 94301, (800) 876-4900, fax (415) 321-3728.

ONE UP CORP. Circle No. 107

SMART 2.0 makes porting feasible. The new version includes an analysis and reporting tool and a source migration tool. It supports 16- and 32-bit Windows ap-

plication migration to 32-bit OS/2 as well as 16-bit OS/2 to 32-bit OS/2. Price: Call (407) 982-6408.

One Up Corp., 1603 LBJ Freewy, Ste. 200, Dallas, Tex. 75234, (800) 678-0187 or (214) 620-1123, fax (214) 620-9626.

**PARCPLACE
SYSTEMS INC. Circle No. 108**

VisualWorks 2.0 is a client/server tool for building applications using object-oriented technology. A Database Application Creator is included for rapid application development. Applications developed are portable across multiple platforms, are scalable across the enterprise, and can have their functionality distributed between both clients and servers. Price: \$4,995.

ParcPlace Systems Inc., 999 East Arques Ave., Sunnyvale, Calif. 94086-4593, (800) 759-7272 or (408) 481-9090, fax (408) 481-9095.

QIIQ LTD. Circle No. 109

Visage 1.00 allows existing DOS programs to be converted into full Presentation Manager applications without requiring any change to the source or executable code. REXX or C is used to produce a graphical front end, which controls the background DOS program. It allows DOS integration for OS/2 applications supporting REXX macros. Price: \$120 to \$450.

Qiiq Ltd., Elm House, 17-19 Claygate Ln., Thames Ditton, Surrey, KT7 0DL, U.K., 0181-339-0739, fax 0181-398-8443.

VISUAL SYSTEMS CORP. Circle No. 110

GUI-Kit 1.0a is a cross-platform GUI toolkit for C and C++. GUI-Kit simplifies the development of GUI-based applications. Once your application is



written, it can be ported to Windows 3.1, Win NT, OS/2, and UNIX/ Motif. Call for a free demo disk. Price: \$249 until Apr. 30, 1995; \$495 thereafter.

Visual Systems Corp., 2512 Crosstown Blvd. NE, Ham Lake, Minn. 55304, (800) 247-2108 or (612) 434-6382, fax (612) 434-6538.

WATCOM INTERNATIONAL CORP.

Circle No. 111

Watcom C/C++ 10.0 is a development system for 32-bit DOS, Windows NT, Win32s, OS/2 2.x, and Novell NLMs, and 16-bit DOS and Windows 3.x. It combines technology with a new integrated development environment (IDE) and set of tools. It includes a GUI debugger, a C++ class browser, and a profiler. There is support for C++ templates, exception handling, and Microsoft Foundation Class libraries (MFC). Price: \$199 for a CD-ROM package.

Watcom VX•REXX Standard Edition 2.1 is a visual development environment for creating graphical OS/2 Presentation Manager applications. VX•REXX combines a project management facility, a visual designer, and an interactive source level debugger to deliver its visual development environment. Price: \$99.

Watcom VX•REXX Client/Server Edition 2.1 is a visual development environment for OS/2. It includes all the features of VX•REXX 2.1 plus connection, query, and chart objects that allow you to access several databases, manipulate data, and chart the results. Features include drag-and-drop programming, bound controls, and multithreaded, multiwindowed, and drag-and-drop-enabled application development. Price: \$299.

Watcom International Corp., 415 Phillip St., Waterloo, Ont., Canada, N2L 3X2, (800) 265-4555 or (519) 886-3700, fax (519) 747-4971.

XVT

SOFTWARE INC.

Circle No. 112

XVT Development Solution for C 4.0 combines XVT-Design with XVT Portability Toolkit. Applications developed with the XVT Development Solution for C will have native look and feel and single-source portability to seven different GUIs and over 30 hardware platforms. Supported GUIs include Windows, Windows NT, Macintosh, OS/2, OSF/Motif for UNIX and VMS, OPEN LOOK, and character screens for DOS, UNIX, and VMS. Price: \$2,325 to \$7,500 per license.

XVT Development Solution for

C++ 3.0 features a true application framework with features such as built-in application document view messaging, automatic data propagation, nested views, drag-and-drop, geometry management, environment inheritance, imaging, and the completely integrated Rogue Wave data structures. Supported GUIs include Microsoft Windows, Windows NT, Macintosh, OS/2, OSF/Motif for UNIX and VMS, OPEN LOOK, and character screens for DOS, UNIX, and VMS. Price: \$2,325 to \$7,500 per license.

XVT Software Inc., 4900 Pearl East Circle, Boulder, Colo. 80301, (800) 678-7988 or (303) 443-4223, fax (303) 443-0969.

ZINC

SOFTWARE INC.

Circle No. 113

Zinc Application Framework 4.0 has a class library and a visual development tool to provide single-source code portability to over 20 platforms. It offers drag-and-drop, geometry management, printing, help, and a variety of user interface objects, such as notebook, toolbar, spinner, and slider. Source code and technical support are included.

Zinc Software Inc., 405 South 100 East, Pleasant Grove, Utah 84062, (800) 638-8665 or (801) 785-8900, fax (801) 785-8996.



New Products for OS/2

BackMaster 1.1. This tape backup software for OS/2 features OS/2 Warp compatibility, bidirectional OS/2 DOS/Windows data interchange via QIC-80 minicartridge tapes, and parallel port tape drive support for Conner, Iomega, Microsolutions, and Tecmar QIC-80 250MB tape drives. In addition, this updated version of BackMaster has an enhanced "boot from floppy" restore utility for rapid restart/recovery, support for high-speed tape accelerator cards for faster backup/restore operation, new command line options, drag-and-drop launch, and custom file save sets.

MSR Corp. Circle No. 151
Phone: (409) 564-1862, Fax: (409) 560-5868

Blinker 3.1. Blinker 3.1 offers incremental linking for protected mode CA-Clipper programs. It provides protected mode support for all Microsoft C/C++ and FORTRAN graphics libraries, allows the creation of Windows DLLs, introduces OS/2 support, offers dual-mode support with internal or external overlay files, and doubles the link time virtual memory capacity for larger libraries and .EXE files. This product requires a minimum of an 8086 microprocessor to link or run real mode programs, while the DOS extender requires a minimum of an 80286 microprocessor to run protected mode pro-

grams and is fully compatible with DPML, VCPI, and XMS specifications. Protected mode programs created by Blinker 3.1 will run under Windows, OS/2, and DOS for maximum portability.

Blinkinc Circle No. 152
Phone: (804) 747-6700, Fax: (804) 747-4200

Clearlook. Developed for OS/2, Clearlook maintains 32-bit power. This multi-threaded application gives users document processing capabilities through implementing object-oriented technology.

Clearlook Corp. Circle No. 153
Phone: (800) 818-LOOK

ENFIN 4.1. This is Easel Corp.'s object-oriented client/server application development environment. ENFIN 4.1 includes enhanced support for reuse of user interfaces through interface components and screen inheritance, an event editor for creating point-and-click interfaces, an integrated programmer's editor for modifying Smalltalk source code, a small program generator to reduce the size of run-time applications, and enhancements for better network communications. Synchronicity 2.1, a business object modeling and persistent object mapping tool has been upgraded to take advantage of the features offered by



ENFIN 4.1. New for ENFIN 4.1 is TCP/IP support for networking across OS/2, Windows, and UNIX. ENFIN 4.1 now includes a class that implements the TCP/IP Application Programming Interface (API) for applications.

Easel Corp. Circle No. 154
Phone: (617) 221-2133

GraphiC/OS2. GraphiC/OS2 is a 32-bit, multithreaded OS/2 application that enables scientists and engineers to make plots of their data without having to learn GUI programming. This C library of over 240 functions may be used to create publication-quality technical graphics. Using a library instead of a stand-alone graphics program allows the user to

see data as they are being generated. GraphiC/OS2 comes with full source code for all its library functions so full customization is possible. This product creates the window into which the graph is drawn and provides menus for postprocessing the plot when it is completed. In addition to managing the window, GraphiC/OS2 provides macros that can obtain prompted user input to display messages and to display textual data in a separate scrolling window. The same example programs for GraphiC work in the DOS, Windows, Windows NT, and OS/2 versions of GraphiC.

Scientific
Endeavors Corp. Circle No. 155
Phone: (800) 998-1571

OS/2 CALENDAR

April 24-27	Comdex/Spring Atlanta, Ga. (617) 449-6600
May 8-10	OS/2 Development '95 Amsterdam, Netherlands 32-2-538-60-40 CompuServe: 76307,1054 Internet: pwesterman@mfi.com
May 21-25	IBM Technical Interchange '95 New Orleans, La. (800) 872-7109 (508) 443-4990
June 20-22	PC Expo New York, N.Y. (800) 829-3976 (201) 346-1400
July 18-21	OS/2 World Boston, Mass. (415) 905-2354
August 14-18	Software Development East Boston, Mass. (800) 441-8826 (415) 905-2784

High C/C++ for OS/2. With DirectToSOM, any C/C++ source code can be compiled with High C/C++ to create SOM-compatible binaries without writing Interface Definition Language class descriptions. SOM objects compiled with High C/C++ for OS/2 can interoperate with SOM objects written in any other language. High C/C++ compilers include fully implemented C++ exception handling (including nested exceptions) so developers can build run-time error handling into applications. Other C++ enhancements include name spaces to avoid name clashes with third-party libraries, run-time type information (RTTI), and ANSI C++ casting notation. This product also provides OMF common areas for templates, virtual function tables, RTTI, and in-line functions. The compiler includes thread-safe libraries that can be used in multithreaded applications, and all library functions are documented as to reentrancy. With High C/C++, developers can apply thread-local storage over large blocks of code or even entire program modules with little or no change to their source code. High C/C++ includes additional switches for fine-tuning function in-lining, additional optimizations to produce executables for specific processors such as the Pentium, improved floating-point performance, and support for long types.

Metaware Inc. Circle No. 156
Phone: (408) 429-6382, Fax: (408) 429-9273

i++. This is a C++ class library for the OS/2 programming interface. The i++ kernel performs reference counting, thereby enabling proper management of class construction and destruction. In addition, this product allows the class library to act as a Virtual C++ operating system. All OS/2 error codes are

mapped to exception codes defined within the exception class hierarchy. The i++ Online Programming Reference extends to one quarter of a million lines and documents all constants, classes, and methods. Over 100 samples covering most aspects of Presentation Manager programming are supplied.

Virtual Software Circle No. 157
Phone: 61 (414) 563-256, Fax: 61 (414) 571-695 (Australia)

Object COBOL 3.3. This is a 32-bit product for migrating existing and developing new COBOL applications that take advantage of high-performance PCs and operating systems as well as the latest programming methodologies. It includes all the features of an object-oriented environment. Object COBOL applications can execute in a native 32-bit environment, provide access to 32-bit API routines, offer 32-bit performance, and provide possible interface to 32-bit non-COBOL programs. Object COBOL also includes ANIMATOR 2, a debugging tool, and support for mixed-language applications.

Micro Focus Circle No. 158
Phone: (415) 856-4161

ODBTalk for OS/2. This product provides the Smalltalk/V developer with full 32-bit ODBC support using the OS/2 Q+E ODBC Driver Manager. The product, together with the appropriate ODBC database drivers, supports all ODBC conformance levels. The Smalltalk developer has access to both low-level classes (ODBC API level) and higher-level classes (connection, statement, query, table, and so on) providing various levels of encapsulations. The product includes on-line help, sample programs, and examples. An ODBC Workbench integrated within the Smalltalk/V for OS/2 development environ-

ment is provided as well. Executables may be distributed with no run-time fees. ODBTalk for OS/2 requires OS/2 2.x, Smalltalk/V for OS/2, and the Q+E Driver Manager and the appropriate database drivers, or the Watcom-SQL database.

LPC Consulting Circle No. 159
Services Phone: (416) 787-5290, Fax: (416) 787-9214

OS/2 Imaging Toolkit. Application developers can incorporate high-performance, faster imaging capabilities for 36 formats into their own products with this software library. The library supports faster image formats: JPEG, TIFF, PCX, DIB, TGA, GIF, WMF, PICT, DCX, WPG, EPS, and BMP. Also added is a color-reduction technology, auto-

matic thumbnails, and compression algorithms.

AccuSoft Corp. Circle No. 160
Phone: (508) 898-2770, Fax: (508) 898-9662

PM Patrol 3.0. This resource management program schedules tasks, monitors and displays operating parameters, and continuously displays selected parameters at the bottom of the computer display screen. From the status line, users have access to detailed system and hardware information, all customizable for every level of OS/2 user. Specific features are included for new and "power" users as well as programmers and LAN administrators. PM Patrol 3.0 minimum system requirements are OS/2 2.1 or OS/2 WARP operating systems.



RDC REXX Diagnostic Commander

REXX Debugging Made EASY!

RDC allows software developers, administrators, and ALL OS/2 professionals to quickly and efficiently code and test REXX command files in a fully interactive environment.

- ♣ Full Screen Display
- ♣ Breakpoints
- ♣ Variable Display Service
- ♣ Variable Alteration
- ♣ Single Step Review
- ♣ Dynamic Watch Windows
- ♣ Logic Alteration
- ♣ Deferred Debug Option

And RDC supports multi-platforms

Suitable Alternatives

4655 Old Ironsides Dr., Suite 220
Santa Clara, CA 95054

**Call (408) 727-3142, (800) 734-7011,
or FAX (408) 727-3170**



MSR Corp. Circle No. 161
Phone: (409) 564-1862

Standard Template Library (STL) <Toolkit>. This product offers multi-thread extensions, including read and write locking, and is compatible with cfront-based compilers. STL <Toolkit> cross-platform availability allows users the choice of Windows, UNIX, or OS/2 environments. STL is a set of reusable collections and algorithms and comes with ThreadKit, ObjectSpace's cross-platform library for multi-thread development. The product includes a comprehensive tutorial and class catalog, samples, test suite, and a commented source code.

ObjectSpace Inc. Circle No. 162
Phone: (214) 934-2496, Fax: (214) 663-3959

SuperCom 3.0. This development tool aids programmers in the development of multiplatform communications applications. The SuperCom libraries allow programmers to develop custom communications applications between computer systems using multiple serial ports in a multitasking environment under OS/2. Specifically, OS/2 developers are able to take advantage of 32-bit OS, multitasking support, and threads support. This allows single threads to be used for simultaneous file transfer. Features for this product include direct register programming; interrupt-driven transmission and reception; line status; modem status; buffering up to 64K; baud rates up to 115 Kbaud; and UART support for 8250, 16450, and 16550. SuperCom is available for C, C++, and Visual Basic.

Fine Line International/
Adontec Computer
Systems GmbH Circle No. 163
Phone: (707) 887-3400, Fax: (707) 887-1015

System Architect. This is a multi-user, repository-based application tool providing automated support for the OMT/Rumbaugh technique. This product provides support for integrated forward and reverse engineering for C++ code generation from designs created using the OMT/Rumbaugh methodology. Both skeleton files and C++ headers for an object-oriented application are automatically generated. System Architect ensures consistency in the manner in which data elements are represented in multiple databases. An optional module for System Architect, known as SA Object-Oriented, supports three object-oriented analysis and design techniques: Booch, Coad/Ed Yourdon, and OMT/Rumbaugh. SA Object-Oriented is \$495. Users of SA Object-Oriented with an active maintenance agreement will receive the OMT/Rumbaugh addition and the upgrades to Booch and Coad/Yourdon without charge. The main System Architect product also includes object-oriented analysis support for Shlaer/Mellor. In early 1995, the Shlaer/Mellor support will be formally added to the object-oriented option. This will include added support for OOD. System Architect runs on any personal computer that supports Windows 3.1 or OS/2 PM 2.1.

Popkin Software &
Systems Inc. Circle No. 164
Phone: (212) 571-3434, Fax: (212) 571-3436

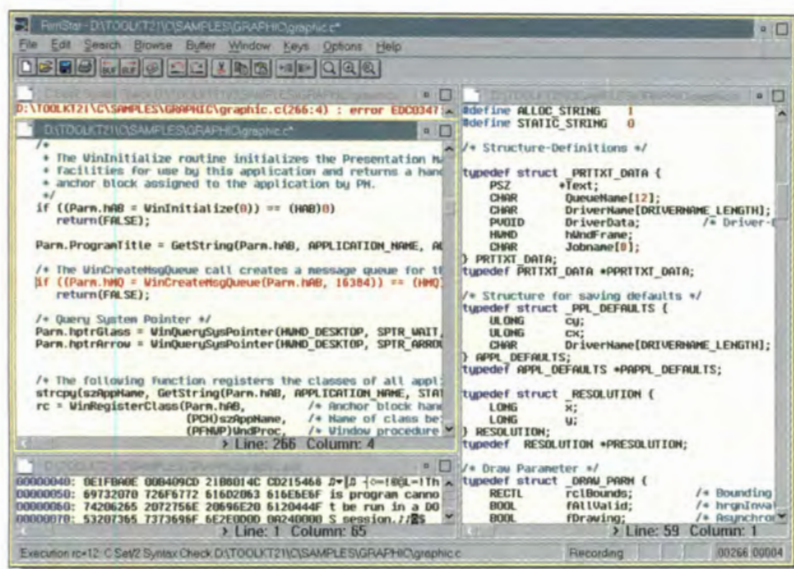
Visual SlickEdit for OS/2. This programmers' editor offers procedure tagging, multiple clipboards, and compiler error processing. It provides the user with a macro language called Slick-C as well as a DLL interface for extending the editor. Visual SlickEdit supports C/C++, Pascal, Fortran, Basic, dBASE, Modula-2, assembly language, COBOL, and Ada.

microEdge Circle No. 165
Phone: (800) 934-EDIT or (919) 831-0600, Fax: (919) 831-0101

XDB-Lite. This SQL database for stand-alone workstations was developed primarily for OEMs, VARS, Distributors, and System Integrators seeking an SQL Engine to embed within their current client/server product offerings. XDB-Lite is a subset of XDB's database engine, featuring a complete implementation of the SQL language as well as many features, including cascading referential integrity, distributed database access, transaction processing, a bidirectional cursor, backup and recovery, password protection, concurrency control, an integrated system catalog, and updatable views. This product also includes DB2 compatibility, a Declaration Generator Utility (DCLGEN), scalable server support, and mainframe access. XDB provides a desktop or LAN-based SQL implementation that is both syntactically and semantically equivalent to IBM's DB2, including SQL Syntax, data types, return codes, EBCDIC support, and system catalog architecture.

XDB Systems Circle No. 166
Phone: (800) 488-4948 or (415) 312-9300 ext. 420

POWER UP WITH THE RIMSTAR™ PROGRAMMER'S EDITOR NEW VERSION 2.1



If you're looking for a fully configurable, professional OS/2 PM programmer's editor to replace your current text-mode editor – we have what you have been looking for!

No hassle changing editors

We know changing editors can be a major hassle. You don't want to relearn a new set of keystrokes no matter how good the product. Well, you don't have to – we have Brief, CUA, Epsilon, Multi-Edit, SlickEdit, PWB, and Borland IDE keyboard mapping waiting for you. If you're not using one of these, let us know and we will create the mappings you need.

Features designed to increase productivity

It has all the features you have come to expect, plus special features designed to anticipate your needs and increase your productivity. Features like a 'C' source browser that eliminates those time consuming searches for functions, global variables, typedefs, and macros by providing you with instant positioning to both definitions and references. Using our powerful 'C' macro language you can customize your programming environment to suit your individual needs.

RimStar Technology, Inc.

91 Halls Mill Road
Newfields, NH 03856
Voice: (603) 778-2500
Fax: (603) 778-2408
BBS: (603) 778-4644

Price \$299.00

Plus Shipping & Handling.

To order call 1-800-746-7007

60 day money-back guarantee.

Also available for Windows and Windows NT
Circle Reader Service Number 23

Partial Feature List

- ✓ Complete ANSI 'C' macro language
- ✓ SyntaColor™—syntax highlighting
- ✓ Smart C/C++ Indenting & brace matching
- ✓ Bookmarks
- ✓ Unlimited undo and redo
- ✓ Timed auto save
- ✓ Access PDK help for function under cursor
- ✓ Multi-threaded for no waiting
- ✓ Compile and jump to errors
- ✓ Import/export to system clipboard
- ✓ Keystroke record/playback
- ✓ Block indent/outdent
- ✓ Hex editing
- ✓ Customizable menus
- ✓ Column, line and block selection, search and replace
- ✓ Integrates with Workframe/2
- ✓ Source browser for 'C'
- ✓ Template editing
- ✓ Multi-buffer regular expression search and replace
- ✓ Support for version control
- ✓ Complete on-line help
- ✓ Save and restore state between sessions
- ✓ OS/2 2.x 32 bit PM Multi-document interface

"My copy of Brief has been permanently retired. Keep up the good work!" - A.L.

All products and company names are trademarks or registered trademarks of their respective holders. RimStar and SyntaColor are trademarks of RimStar Technology, Inc.

© 1994 RimStar Technology Inc.

In the Race for Solutions, Finish First



Introducing VisPro/Reports, the first programmable report writer for VisPro/REXX, VX REXX and OS/2 REXX

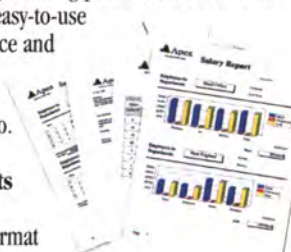
On April 28, 1993 HockWare Inc. shipped VisPro/REXX, the first OS/2 visual programming tool for REXX. Since then, we've achieved many other firsts. VisPro/REXX, VisPro/C and VisPro/C++ are the first REXX, C and C++ tools to incorporate drag and drop programming, an entity-relationship database designer, and a rich set of add-on objects including: spreadsheet, business graphics, formatted entry field, clock and calendar. Whether you're using REXX, C-Presentation Manager, or IBM's User-Interface Class Library, there is a VisPro product to meet your needs.

Yet Another First

VisPro/Reports is the latest addition to the VisPro family of OS/2 programming products. VisPro users will recognize the easy-to-use CUA'91 user-interface and the drag and drop style HockWare pioneered years ago.

Designing Reports is Easy

- WYSIWYG, free-format report design tool.



- Banded report options allow you to choose from a rich set of headers, footers and groups.
- Label reports allow you to select from a large list of Avery labels.
- Support for multiple line text, derived fields, business graphics, bitmap images, rectangles, lines and ellipses.
- Many object properties to choose from, including any OS/2 font, 16 million colors, line styles, fill patterns and drop shadows.
- Wide selection of field options, International currency and formulas.
- Numerous sample reports included.

Report Printing is Easy

- Seamlessly print or preview reports from any OS/2 REXX environment including VisPro/REXX, VX REXX and plain OS/2 REXX.
- Use your favorite REXX-enabled databases and third party libraries for access to DB2, .DBF files, EHLLAPI mainframe connections and more.
- A runtime DLL of less than 100K can be distributed royalty free.

The Complete VisPro Family

VisPro/Reports	\$199
VisPro/REXX Gold	\$299
VisPro/REXX Bronze	\$ 99
VisPro/REXX Data Entry Object Pack.	\$119
VisPro/C or VisPro/C++	\$399
VisPro Development Suite	\$599
(includes VisPro/REXX Gold, C and C++)	

Whether you've already discovered the VisPro family of visual programming products or you're just getting to know us, meet our latest member, VisPro/Reports. And be the first to get where you're going.

Special Offer
Buy
VisPro/REXX Gold
for \$299 and get
VisPro/Reports
absolutely free.
Offer Expires 4/15/95



HockWare Incorporated
P.O. Box 336
Cary, NC 27512-0336
919-380-0616
919-380-0757 FAX
Go HockWare on CompuServe
hockware@vnet.net on Internet

HockWare, VisPro/C, VisPro/C++, VisPro/REXX and VisPro/Reports are trademarks of HockWare Incorporated. All other company, product and brand names are trademarks and/or registered trademarks of their respective holders and are mentioned for reference purposes only. ©1995 HockWare Incorporated. All rights reserved.

Circle Reader Service Number 3